2021 혁신성장 청년인재 집중양성 추경사업

Python 기반 AI를 활용한 빅데이터 분석가 양성 과정

머신러닝을 위한 Python 라이브러리

2021. 12. 01. ~ 2021. 12. 16.

메디치교육센터

과목 소개

개요

빅데이터 분석을 위한 머신러닝 기법들을 학습하고, Python에서 제공하는 Library를 가지고 이러한 기법들을 구현한다.
구현을 통해 각 기법들의 특징과 원리를 이해하고 실제 데이터 분석을 실습하여 실무에 적용하는 방법을 익힌다.

교육 목표 및 기대효과

- 빅데이터 분석에 필요한 다양한 머신러닝 개념을 이해하고 실무에 적용하는 방법들을 학습한다.
- 머신러닝 기법들을 Python Library를 활용하여 구현하는 능력을 갖춘다.
- 여러 분야의 빅데이터들에 대해 머신러닝을 적용하는 실습을 통해 머신러닝을 활용한 분석 능력을 향상시킨다.

과목 구성

머신러닝의 이해

머신러닝의 개요
필수 라이브러리
퍼셉트론 알고리즘

머신러닝의 구현(I)

사이킷런의 이해
KNN 알고리즘
선형회귀 알고리즘
로지스틱회귀 알고리즘

머신러닝의 구현(II)

데이터 전처리
누락 데이터 다루기
사이킷런 추정 API
범주형 데이터 다루기
훈련vs.테스트 데이터셋

머신러닝의 구현(III)

의사 결정트리 알고리즘
나이브 베이즈 알고리즘
군집화(클러스터) 기법
앙상블 학습 기법

실무 적용

예측 프로젝트
추천 프로젝트

CONTENTS

CONTENTS

일차	주제	비고
1일차	머신러닝의 이해	
2일차	데이터 전처리, KNN 알고리즘의 이해	미니프로젝트 (코로나/경찰청/기상청)
3일차	회귀모델(KNN회귀, 선형)	
4일차	선형회귀(다중회귀, 리지, 라쏘)	
5일차	로지스틱 회귀(예제: 보스턴집값)	미니프로젝트 (심장질환/ 암진단/버스정류장 묶음과 파출소설치)
6일차	의사결정트리1	
7일차	의사결정트리2(예제: 클릭예측)	
8일차	나이브베이즈(스팸메일분류기)	
9일차	군집화	
10일차	앙상블	
11일차	예측 분석 관련 알고리즘	
12일차	추천 관련 알고리즘	

시작하기 전에..

1. 환경설정
2. 머신러닝 용어
3. 머신러닝의 작업흐름

1. 환경설정

1 머신러닝을 위한 파이썬

- Python 버전 3.7.2 기준(버전 3.7.2 이상 제공)
- Pip 명령으로 파이썬 패키지를 설치하여 사용

  > pip install 패키지 이름(패키지 새로 설치)

  > pip install –upgrade 패키지이름(설치한 패키지 업데이트)

2 파이썬 배포판과 패키지 관리를 위한 아나콘다

- 아나콘다 설치(https://www.anaconda.com/download)
- 아나콘다 설치 후 파이썬 패키지를 설치할 수 있다.

  > conda install 패키지 이름(패키지 새로 설치)

  > conda update 패키지이름(설치한 패키지 업데이트)

3 그 외에 머신러닝을 패키지 버전들

- NumPy 1.19.5
- Scipy 1.4.1
- Scikit-learn 0.23.2
- Pandas 1.1.5

4 conda 환경설정 + jupyter notebook 실행

- conda info : conda 환경에 대한 자세한 정보를 조회
- conda에 새로운 환경을 생성
  - conda create –n **ML_py** python=3.9.7 **anaconda** : create 옵션을 사용하여 새로운 환경 생성

    → ML_py (이름을 자유로 붙임) / anaconda 패키지 리스트를 쓰면 제공하는 라이브러리 모두 설치됨
  - conda activate ML_py : 새로 생성한 conda 환경을 활성화 시킴
- Jupyter Notebook의 설정 파일을 생성
  - jupyter notebook –generate-config
- Jupyter Notebook의 실행
  - jupyter notebook

5 jupyter notebook 단축키

- 명령모드에서 셀 실행
  - Ctrl + Enter(현재 셀 실행)
  - Shift + Enter(현재 셀 실행 후 아래 셀로 이동)
  - Alt + Enter(실행 후 아래에 셀 추가)
- 명령모드에서 셀 편집 단축키
  - A(현재 셀 위에 셀 추가)
  - B(현재 셀 아래 셀 추가)
  - DD(현재 셀 삭제)
  - C(선택한 셀 복사)
  - V(붙여넣기)

6 구글 코랩 사용하기

- 준비물
  - 구글 계정
  - 웹브라우저(가능하면 크롬)
- 접속하기/ 시작하기
  - https://colab.research.google.com/
- 실행하기
  - 텍스트 셀: Cell은 코랩에서 실행할 수 있는 최소 단위이다.
  - 텍스트 셀에 대한 제약사항이 적다.
  - 코드셀: 코드 셀로 이동하면, 코드와 명령을 수행한 결과가 함께 표신된다.
  - 확장자:ipynb 이고, 노트북(코랩 노트북)이라고 부른다.
- 구글 드라이브 마운트
  - UI를 사용하여 마운트 하거나, 직접 커맨드를 사용하여 마운트 가능
- 시스템 스펙 확인
  - CPU: $cat /proc/cpuinfo
  - Memory: $cat /proc/meminfo

2. 머신러닝 용어

◆ **훈련샘플**: 데이터셋을 나타내는 테이블의 행

> **동의어**: 관측(Observation), 레코드(Record), 인스턴스(instance), 예시(example)

◆ **훈련**: 모델피팅, 모수 모델의 경우 파라미터 추정과 비슷

◆ **특성(x)**: 데이터 테이블이나 데이터 행렬의 열,

> **동의어**: 예측변수(predictor variable), 변수, 입력, 속성(attribute), 공변량(covariate)

◆ **타깃(y)**

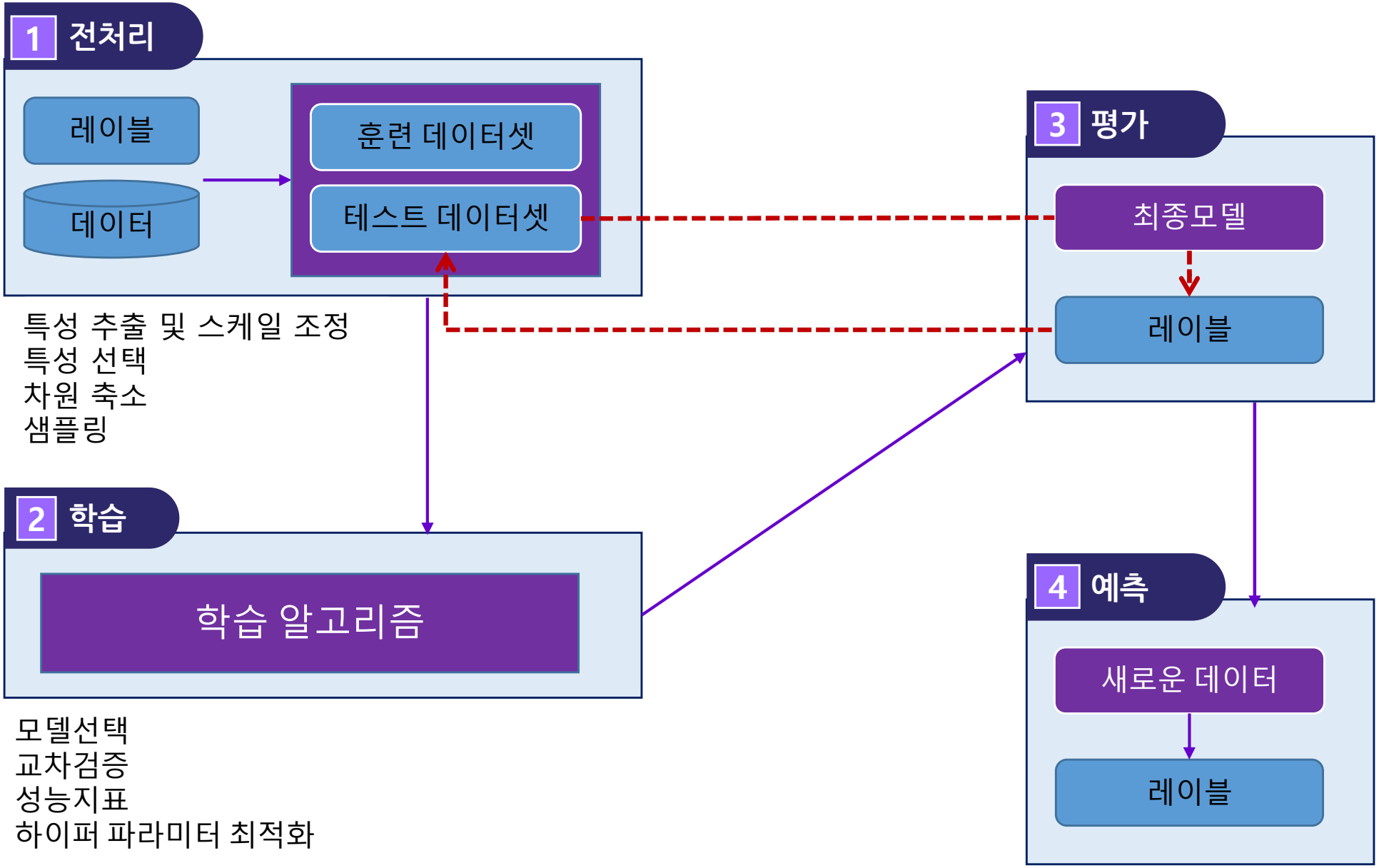
> **동의어**: 결과(outcome), 출력(output), 반응 변수, 종속 변수(dependent variable),
(클래스)레이블(label), 정답(ground truth)

◆ **손실 함수(loss function):** 종종 비용 함수(cost function)와 동의어로 사용

일부 자료에서는 손실 함수를 하나의 데이터 포인트에 대해 측정한 손실로 사용

비용함수는 전체 데이터셋에 대해 계산한 손실로 사용한다.

3. 머신러닝의 작업 흐름

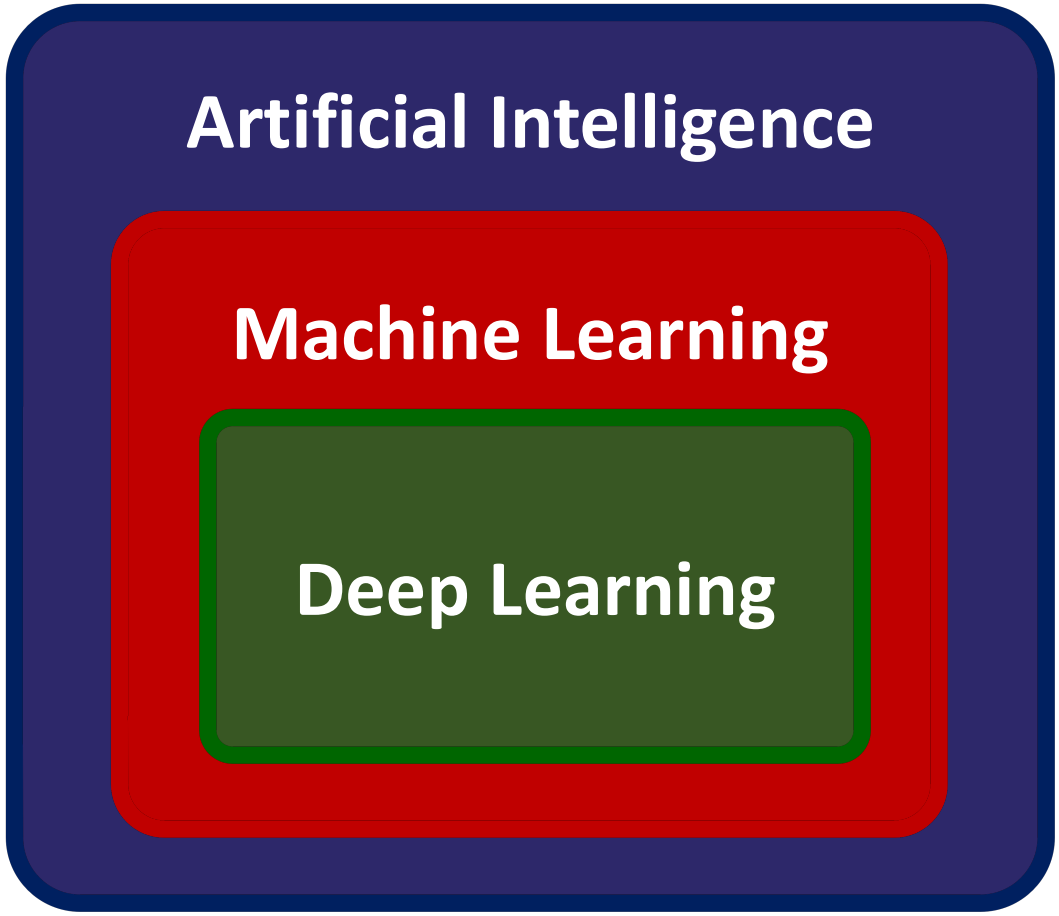

1 전처리
레이블
데이터
훈련 데이터셋
테스트 데이터셋
특성 추출 및 스케일 조정
특성 선택
차원 축소
샘플링
2 학습
학습 알고리즘
모델선택
교차검증
성능지표
하이퍼 파라미터 최적화
3 평가
최종모델
레이블
4 예측
새로운 데이터
레이블

I-1. 머신러닝의 개요

1. 머신러닝이란?
2. 머신러닝을 사용하는 이유
3. 머신러닝 학습종류

머신러닝이란

- 기계가 학습을 할 수 있도록 하는 연구분야
- 인공지능 연구의 한 분야로서 최근 들어 딥러닝을 통해서 빠르게 발전



Artificial Intelligence

Any technique which enables computers to mimic human behavior.

(인간과 비슷하게 사고하는 컴퓨터 지능 포괄)

Machine Learning

Subset of AI techniques which use statistical methods to enable machines to improve with experiences.

(데이터를 입력해 컴퓨터를 학습시키거나 스스로 배우게 해서 인공지능 성능을 향상시키는 방법)

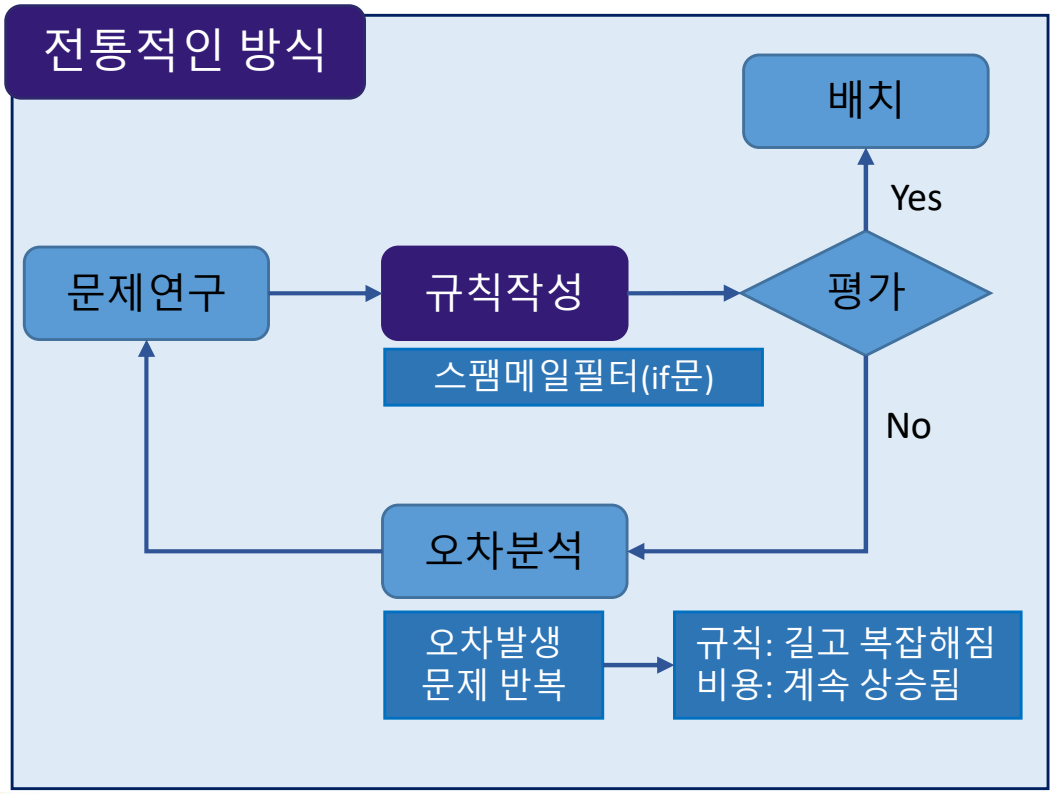
Deep Learning

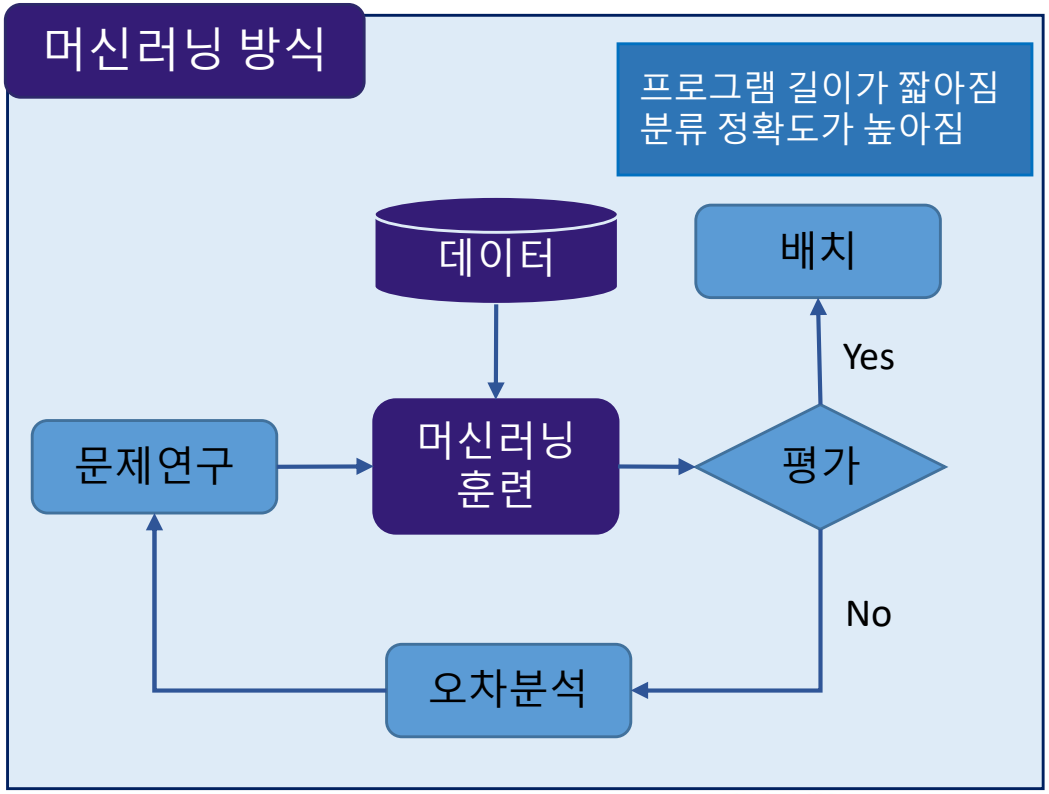
Subset of ML which make the computation of multi-layer neural networks feasible.

(인공신경망 이론을 기반으로 복잡한 비선형 문제를 기계가 스스로 학습 해결)

1 머신러닝의 개념

- 데이터로부터 학습하도록 컴퓨터를 프로그래밍하는 과학
- 명시적인 프로그래밍 없이 컴퓨터가 스스로 학습하는 능력을 갖게 하는 연구분야
- 과거 경험에서 학습을 통해 얻은 지식을 미래의 결정에 이용하는 컴퓨터 과학의 한 분야
- 관측된 패턴을 일반화하거나 주어진 샘플을 통해 새로운 규칙을 생성하는 것이 목표



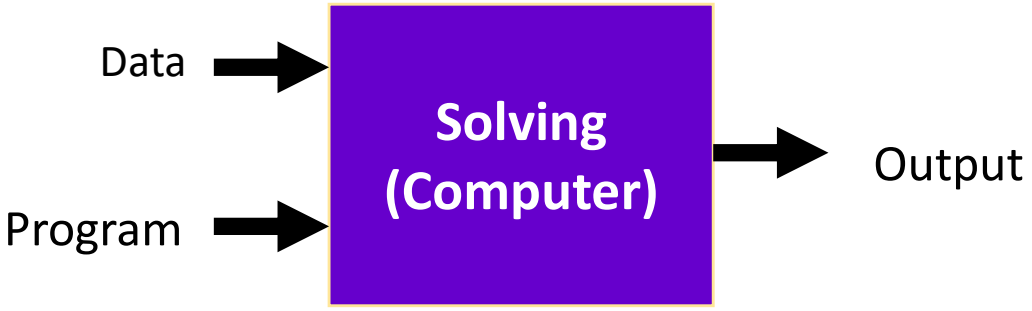


머신러닝이란

1 프로그래밍 방식 vs. 머신러닝 방식

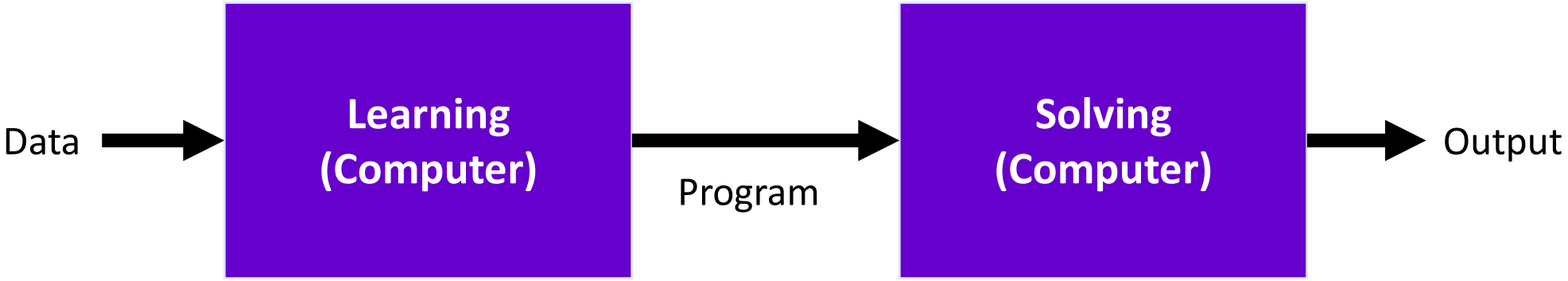
일반적인 Programming

- 사람이 알고리즘 설계 및 코딩
- 주어진 문제(데이터)에 대한 답 출력



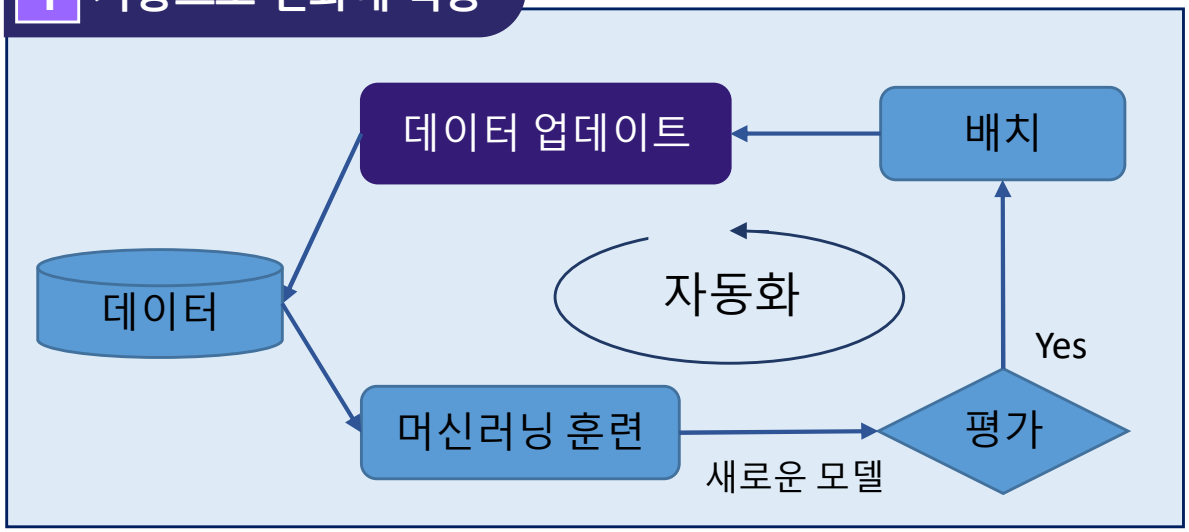
머신러닝 Programming

- 사람이 코딩
- 기계가 알고리즘을 자동 프로그래밍(Automatic Programming)
- 데이터에 대한 프로그램을 출력

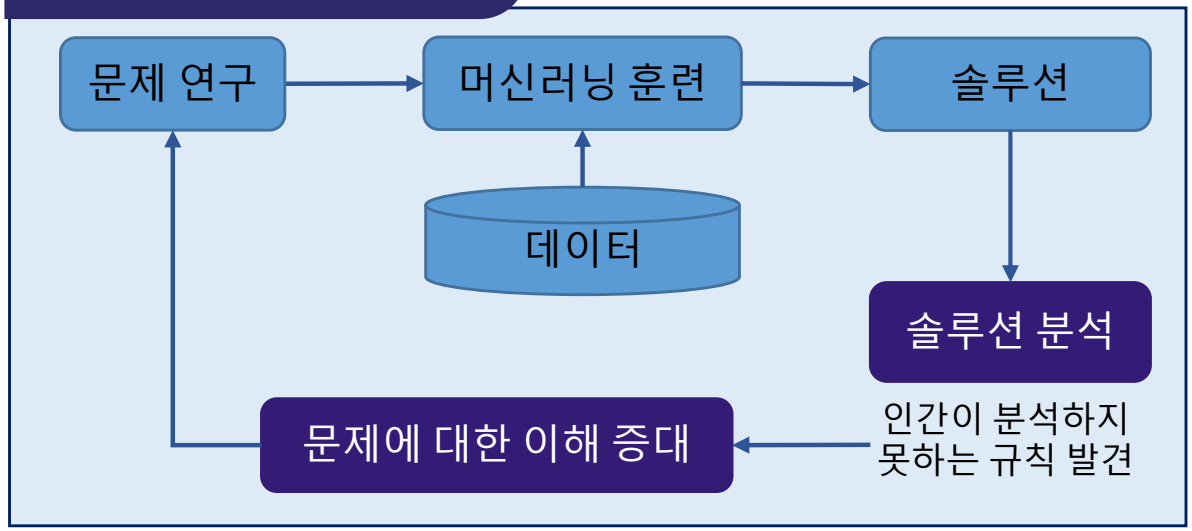


2. 머신러닝을 사용하는 이유

1 자동으로 변화에 적응



2 새로운 패턴을 발견



3. 머신러닝 학습 종류

1 머신러닝의 학습 종류

3. 머신러닝 학습종류-지도학습

1 지도학습

- 학습 데이터가 입력(특징 행렬)과 출력(대상 벡터) 쌍으로 제공됨 → "레이블 데이터"
- 학습 목표는 입력 특징 행렬과 출력 대상 벡터를 매핑시키는 규칙을 찾는 것
- 입력 특징 행렬에 대해 출력 대상 벡터가 알려져 있으므로 "지도"라 불림

2 해결할 수 있는 대표적인 문제

회귀(Regression)	분류(Classification)
연속형 수치 데이터 예측 집값, 중고차 가격, 주가 예측 등	범주형 데이터인 클래스 레이블 예측 스팸 메일 필터, 긍정/부정의 감정분석등

- K-최근접 이웃, 선형회귀, 로지스틱 회귀, 서포트 벡터 머신, 결정 트리와 랜덤 포레스트, 신경망

3. 머신러닝 학습종류-비지도학습

1 비지도학습

- ◆ 학습 데이터로 입력(특징 행렬)만 제공됨 → "레이블 없는 데이터"
- ◆ 입력 특징 행렬에 대한 출력 대상 벡터가 없으므로 "비지도"라 부름

2 해결할 수 있는 대표적인 문제

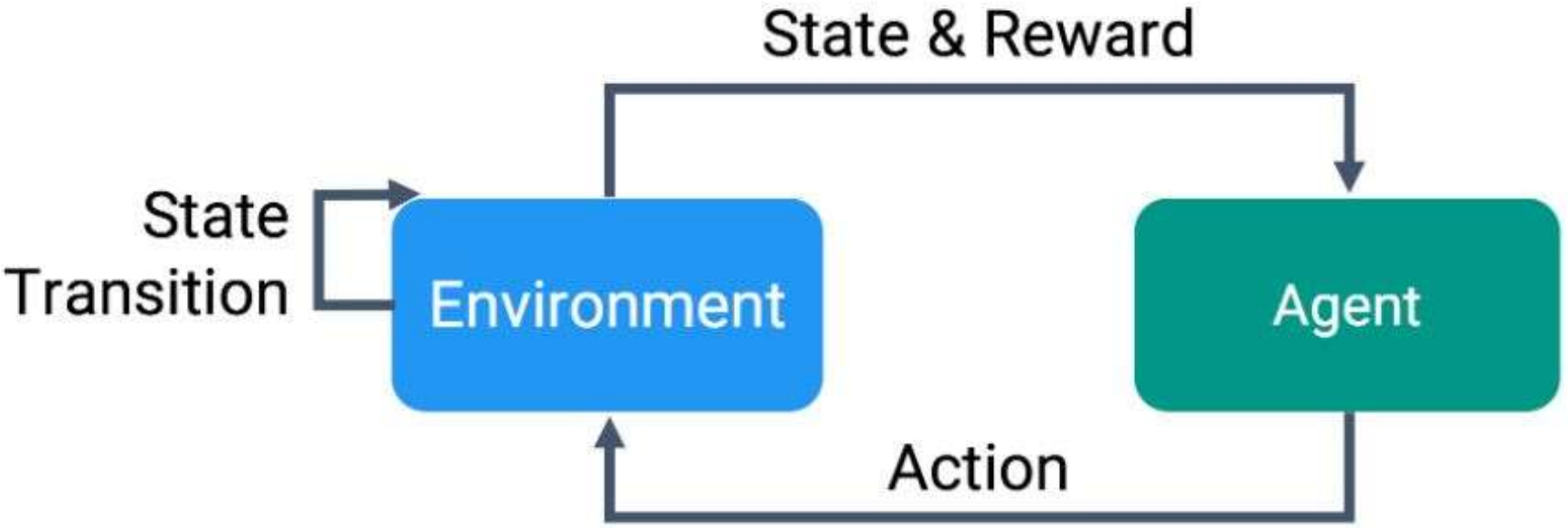
군집화(Clustering)	차원 축소(Demesion Reduction)	연관 규칙학습(Association Rule)
특징이 비슷한 것들끼리 묶어 군을 만드는 것 K-Mean, 계층 군집 분석(HCA), 기대값 최대화 등	인간이 인지할 수 있는 차원(2차원 등)으로 축소하는 것 주성분 분석(PCA), 커널 PCA, 지역적 선형 임베딩(LLE), t-SNE 등	구매 경향성 규칙 발견 등 (장바구니 분석) Apriori, Eclat

- ◆ 이외에도 추천 엔진 등이 있다.

3. 머신러닝 학습종류-강화학습

1 강화학습

- ◆ 시스템이 어떤 목표를 달성하기 위해 동적인 조건에 적응하도록 하는 학습 방식
- ◆ 학습하는 시스템(에이전트)이 환경 관찰 → 액션 실행 → "보상" 또는 "벌점"
- ◆ 시간이 경과하면서 가장 큰 보상을 얻기 위해 최상의 전략("정책")을 스스로 학습
- ◆ 자율 주행 자동차, 알파고 등이 있음



3. 머신러닝 학습종류-준지도학습

1 준지도 학습

- 학습 데이터에 레이블이 일부만 있는 경우 활용
- 데이터 세트 전체에 레이블을 붙이는 데 고비용이 발행
- 지도학습과 비지도 학습의 조합으로 이루어 짐
- 심층 신뢰신경망(DBN)은 제한된 볼츠만 머신(RBM)과 같은 비지도 학습에 기초
- 사진 입력을 통한 사람 식별(군집) + 레이블 값

I-2. 퍼셉트론 알고리즘

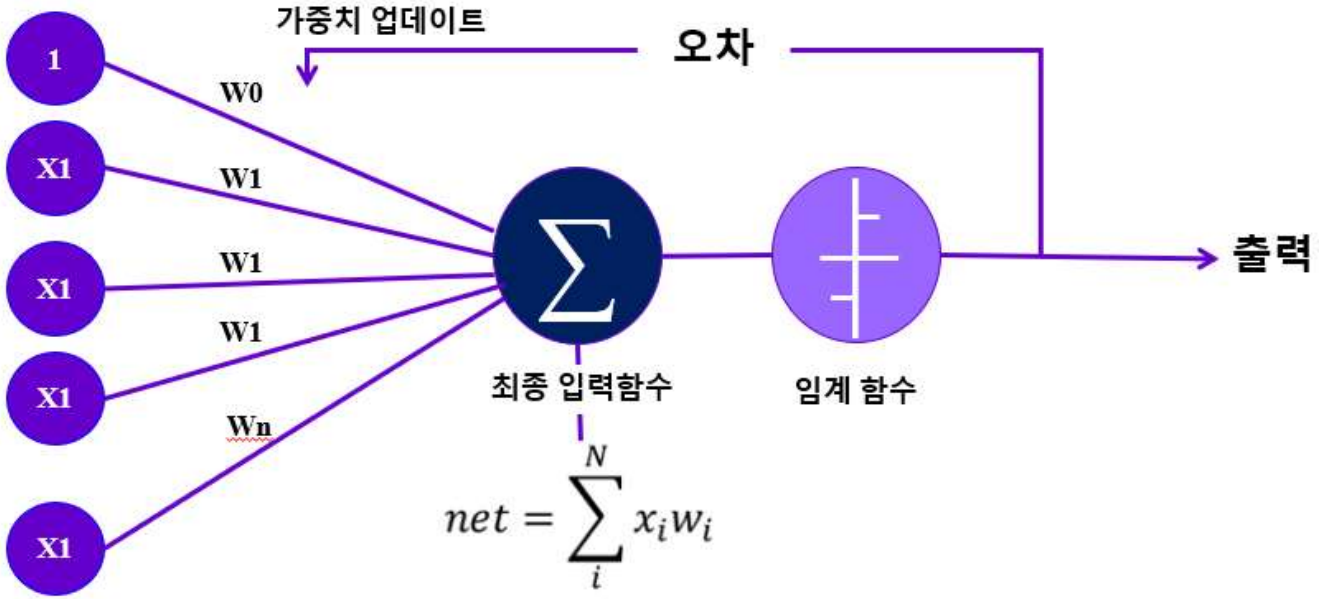
1. 퍼셉트론 구조와 동작
2. 퍼셉트론 학습 규칙
3. 퍼셉트론 실습

1. 퍼셉트론 구조와 동작

1 퍼셉트론이란?

- 신경망의 기원이 되는 알고리즘으로 다수의 신호(int(t):x1, x2,..)를 입력으로 받아 하나의 신호(out(t))를 출력
- 퍼셉트론은 복수의 입력 신호에 고유한 가중치(w1, w2,..)를 부여

  → 각 신호가 결과에 주는 영향력을 조절하는 요소로 작용
- 퍼셉트론의 동작 과정



W

주어진 데이터를 기반으로 학습하여 최적의 W(weight)값을 찾을 때 까지 반복적인 연산을 수행, 점차적으로 W의 값을 보정해 나가는 것

1 퍼셉트론 학습

- 퍼셉트론 학습이란, 훈련 집합이 주어졌을 때 주어진 훈련 집합을 옳게 분류하는 퍼셉트론을 찾는 것
- 알고리즘
  - 단계1: 분류기 구조를 정의하고 분류과정을 수학식으로 표현

Step 1

$$\theta = \{w, b\}$$

  - 단계2: 분류기의 품질을 측정할 수 있는 비용 함수를 정의한다.

Step 2

$$J(\theta) = \sum_{x_k \in Y} (-t_k)(w^T x_k + b)$$

  - 단계3: 비용 함수를 최대 또는 최소로 하는 파라미터 θ를 찾기 위한 알고리즘을 설계

Step 3

$J(\theta) = 0$ 인 θ 찾기 ⟹ SGD (Gradient decent method)
- 초기 해 설정
- 멈춤 조건이 만족될 때 까지 현재 해를 $-\partial/\partial\theta$ 방향으로 이동
- 학습률 ρ를 곱하여 조금씩 이동

II-1.사이킷런의 이해

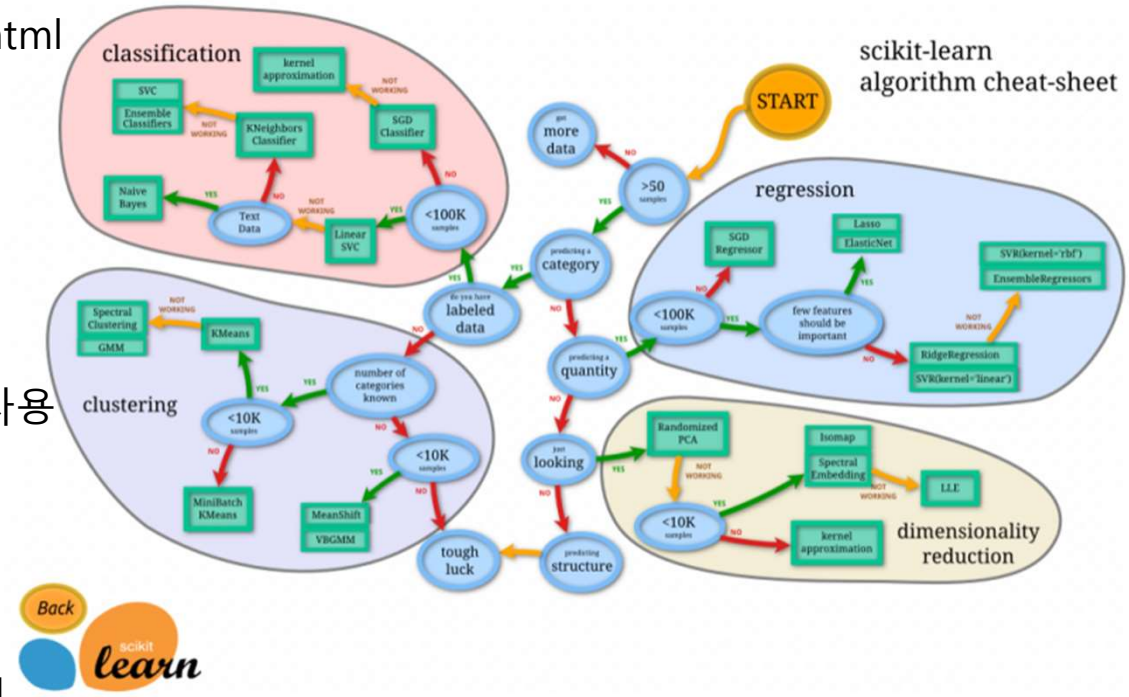
1. Scikit-Learn 라이브러리

2. Scikit-Learn 제공 API 활용방법

1. Scikit-Learn 라이브러리

1 개요

- 머신러닝 알고리즘을 구현한 오픈소스 라이브러리 중 가장 유명한 라이브러리
- 일관되고 간결한 API가 강점, 여러가지 머신러닝 모듈로 구성됨, 문서화가 잘 되어 있음
- 알고리즘은 파이썬 클래스로 구현되고, 데이터 셋은 Numpy array, Pandas DataFrame, Scipy 희소행렬을 사용
- 기본적인 형태로 알고리즘은 라이브러리 import & 모델 생성, 피팅, 그후 예측하는 과정으로 정리됨.
- https://scikit-learn.org/stable/tutorial/basic/tutorial.html



2 데이터 표현 방식

- [n_samples, n_features] 형태의 2차원 배열 구조를 사용
  - Samples: 행의 개수
  - Features: 열의 개수
- 데이터의 형태를 사이킷런이 사용하는 형태로 변형이 필요할 때 numpy의 reshape을 사용하여 변형
  - X=x.reshape(-1,1) 사용

2. Scikit-Learn 자주 사용되는 알고리즘

1 Linear Regression

◆ Import Library & Create Model

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
```

◆ Fit

```
LR.fit(xtrain_data, ytrain_data)
```

✓ LR.coef_ : 기울기

✓ LR.intercept : y절편

◆ Predict

```
predictions = LR.predict(test data)
```

✓ .score() : 결정계수 반환

2 Estimator 기본 활용

◆ 회귀 모델의 성능 평가 방법: 선형회귀

:오차, 에러가 작을 수록, 내가 예측한 결과와 비슷할 수록 좋은 모델

:예측한 결과와 크게 차이가 없는 모델이 좋은 모델이다.

◆ 잔차분석

:잔차 그래프를 통해 실제 값과 예측 값 사이의 차이를 알 수 있다.

◆ 평균제곱오차계산

:평균 제곱 오차를 확인하는 방법

:선형 회귀 모델을 훈련하기 위해서 최소화하는 제곱 오차 항의 평균을 의미

:주로 그리드 서치와 교차 검증에서 매개변수를 튜닝하거나

:다른 회귀 모델을 비교할 때 유용하다.

2. Scikit-Learn 자주 사용되는 알고리즘

3 Naive Bayes

- Import Library & Create Model

```
from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
```

- Fit

```
NB.fit(xtrain_data, ytrain_data)
```

- Predict

```
predictions = NB.predict(test data)
probabilities = NB.predict_proba
```

4 K-NearestNeighbors

- Import Library & Create Model

```
from sklearn.neighors import KNeighborsClassfier
kn = MultinomialNB()
```

- Fit

```
NB.fit(xtrain_data, ytrain_data)
```

- Predict

```
predictions = NB.predict(test data)
probabilities = NB.predict_proba
```

2. Scikit-Learn 자주 사용되는 알고리즘

5 K-Means

- Import Library & Create Model

  from sklearn.cluster import KMeans

  KM = KMeans(n_clusters=4, init='random')

  - n_clusters: 형성할 군집 수와 생성할 중심 수
  - init: 초기화 메소드
    - ✓ k-means++:K-Means++
    - ✓ random:K-Means
  - random_state:임의의 수를 재연할 때 쓰는 값
- Fit

  KM.fit(xtrain_data)
- Predict

  predictions = KM.predict(xtest data)

II-2. K-NN알고리즘

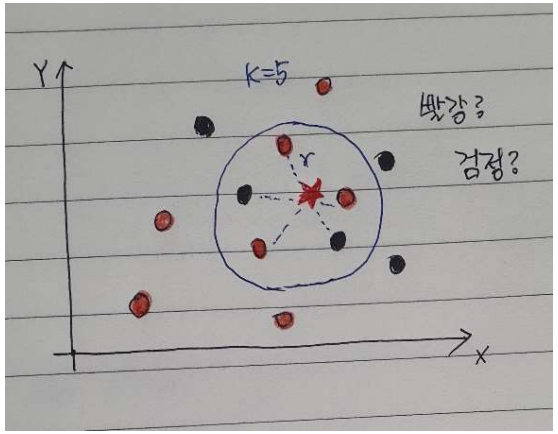
1. K-NN 알고리즘

2. K-NN 예제

1. K-NN(K-최근접이웃)알고리즘

1 개요 및 원리

- 지도학습 알고리즘
- 주변에서 가장 가까운 이웃 K(기본값 5)개의 데이터를 보고, 다수결의 원칙에 따라 주어진 값을 분류
- K는 보통 홀수로 하나, 짝수의 경우 결정이 나지 않으면, 결정 값이 나올 때 까지 K를 줄인다.



1 장단점

- 장점
  - 이해하기가 쉽고 조정 없이도 좋은 성능을 내는 모델이다.
  - 분류와 회귀에 모두 적용 가능하다.
- 단점
  - 데이터가 많아지면 모든 데이터에 대해서 거리를 구해야 하기 때문에 계산 량이 많아져서 시간이 오래 걸림
    ➔ 예측이 느려짐
  - 수 백개 이상의 많은 특성을 가진 데이터 세트와 특성 값 대부분이 0인 희소 데이터 셋에는 작동이 어렵다
  - 분산이 큰 데이터에 성능이 떨어지기 때문에 스케일링이 필요하다.
  - 전처리 과정이 중요하고 성능에 영향을 많이 미친다.

1 분류 예제 1

◆ 도미와 빙어를 분류하자.

K-NN 최근접 이웃 알고리즘

In [32]:
```
#데이터 준비
```

In [33]:
```
#도미 길이&무게 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

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

#빙어 길이&무게 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]
```

In [34]:
```
print(bream_length, bream_weight)
```

```
[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] [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]
```

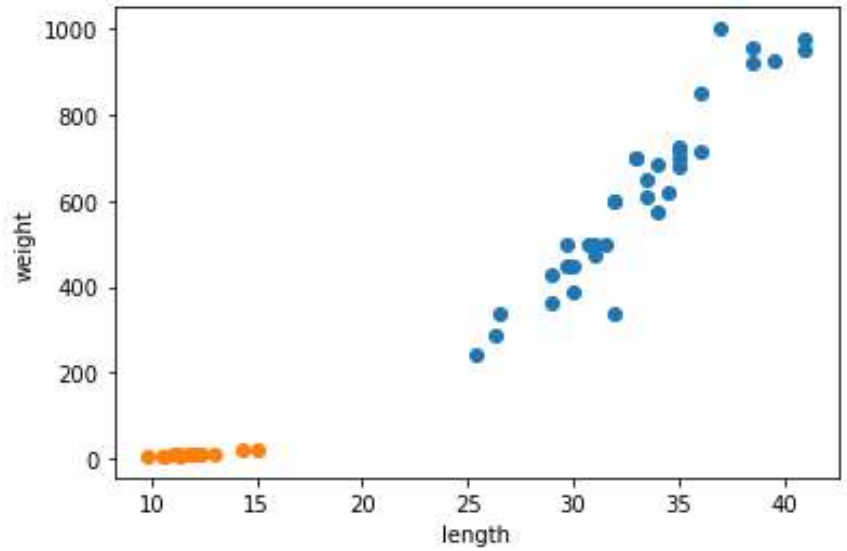
In [35]:
```
print(smelt_length, smelt_weight)
```

```
[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] [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]
```

2. K-NN예제

In [36]:
```
import matplotlib.pyplot as plt
```

In [37]:
```
#화면에 찍어보기
plt.scatter(bream_length, bream_weight)
plt.scatter(smelt_length, smelt_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



In [38]:
```
#사이킷런에서 사용하기 위한 데이터 형태로 변형
#하나의 데이터 셋으로 합치기
length = bream_length + smelt_length
weight = bream_weight + smelt_weight
```

In [39]:
```
print(length, weight)
```

```
[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, 9.8, 1
```

2. K-NN예제


```
In [43]: #지도학습을 위한 정답데이터 만들기(정답데이터는 target이라 부름)
         fish_target = [1]*35 + [0]*14
         print(fish_target)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
In [44]: #사이킷런에서 제공되는 K-NN 알고리즘 사용 임포트
         from sklearn.neighbors import KNeighborsClassifier
         import sklearn
         kn = KNeighborsClassifier()
```

```
In [45]: #학습시키기
         kn.fit(fish_data, fish_target)
```

```
Out[45]: KNeighborsClassifier()
```

```
In [46]: #학습된 모델 점수
         kn.score(fish_data, fish_target)
```

```
Out[46]: 1.0
```

```
In [47]: #학습된 모델로 예측하기
         kn.predict([[30, 600]])
```

```
Out[47]: array([1])
```

```
In [48]: #하이퍼파라미터 k값 조절하기
         from sklearn.neighbors import KNeighborsClassifier
         import sklearn
         kn49 = KNeighborsClassifier(n_neighbors=49) #주변 비교 데이터를 49개, 전체로 사용
```

```
In [49]: #전체 데이터를 비교 범위로 놓고 테스트한 결과
         kn49.fit(fish_data, fish_target)
         kn49.score(fish_data, fish_target)
```

```
Out[49]: 0.7142857142857143
```

```
In [50]: #전체 데이터에서 맞출경우를 단순하게 계산한 값
         print(35/49)
```

```
0.7142857142857143
```

2 분류 예제 2

◆ 여러가지 생선을 분류하자. 앞선 예제에 2가지 생선을 추가 하였다.

In [54]:
```
#추가 생선1: 농어의 길이와 무게 데이터 추가
perch_length = [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
perch_weight = [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,
```

In [69]:
```
print(perch_length, perch_weight)
#정답데이터를 만들기 위해 총 몇개의 데이터를 추가했는지 확인
print(len(perch_length))
```

```
[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] [5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0, 115.0, 12
5.0, 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, 110
0.0, 1000.0, 1100.0, 1000.0, 1000.0]
56
```

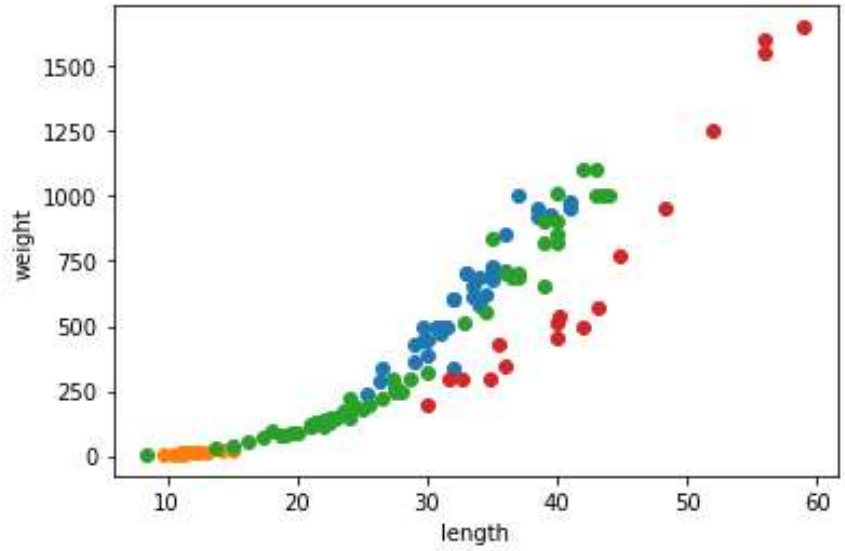
In [58]:
```
#추가 생선2: 강꼬치의 길이와 무게 데이터 추가
Pike_length = [30, 31.7, 32.7, 34.8, 35.5, 36, 40, 40, 40.1, 42, 43.2, 44.8, 48.3, 52, 56, 56, 59]
Pike_weight = [200, 300, 300, 300, 430, 345, 456, 510, 540, 500, 567, 770, 950, 1250, 1600, 1550, 1650]
```

In [68]:
```
print(Pike_length, Pike_weight)
#정답데이터를 만들기 위해 총 몇개의 데이터를 추가했는지 확인
print(len(Pike_length))
```

```
[30, 31.7, 32.7, 34.8, 35.5, 36, 40, 40, 40.1, 42, 43.2, 44.8, 48.3, 52, 56, 56, 59] [200, 300, 300, 300, 430, 345, 456, 510, 540, 500, 56
7, 770, 950, 1250, 1600, 1550, 1650]
17
```

2. K-NN예제

In [61]:
```
#화면에 찍어보기
#주황색:smelt / 파랑색:bream / 초록색:perch / 빨간색:Pike
plt.scatter(bream_length, bream_weight)
plt.scatter(smelt_length, smelt_weight)
plt.scatter(perch_length, perch_weight)
plt.scatter(Pike_length, Pike_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



2. K-NN예제

In [70]:
```
#사이킷런에서 사용하기 위한 데이터 형태로 변형
#하나의 데이터 셋으로 합치기
length = smelt_length + bream_length + perch_length + Pike_length
weight = smelt_weight + bream_weight + perch_weight + Pike_weight
```

In [71]:
```
print(length, weight)
```

```
[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, 25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7
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
39.5, 41.0, 41.0, 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,
2.5, 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,
7.0, 39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0, 43.0, 43.5, 44.0, 30, 31.7, 32.7, 34.8, 35.5, 36, 40, 40, 40.1, 4
48.3, 52, 56, 56, 59] [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, 242.0, 290.0, 340.0, 363.
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, 68
5.0, 720.0, 714.0, 850.0, 1000.0, 920.0, 955.0, 925.0, 975.0, 950.0, 5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 8
5.0, 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,
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
0, 1100.0, 1000.0, 1100.0, 1000.0, 1000.0, 200, 300, 300, 300, 430, 345, 456, 510, 540, 500, 567, 770, 950, 1250, 1600, 1550
```

In [72]:
```
#데이터구조 바꾸기, 리스트의 리스트 형태
fish_data = [[l,w] for l,w in zip(length, weight)]
```

In [73]:
```
print(fish_data)
```

```
[[9.8, 6.7], [10.5, 7.5], [10.6, 7.0], [11.0, 9.7], [11.2, 9.8], [11.3, 8.7], [11.8, 10.0], [11.8, 9.9], [12.0, 9.8], [12.2,
13.4], [13.0, 12.2], [14.3, 19.7], [15.0, 19.9], [25.4, 242.0], [26.3, 290.0], [26.5, 340.0], [29.0, 363.0], [29.0, 430.0],
[29.7, 500.0], [30.0, 390.0], [30.0, 450.0], [30.7, 500.0], [31.0, 475.0], [31.0, 500.0], [31.5, 500.0], [32.0, 340.0], [32.
2.0, 600.0], [33.0, 700.0], [33.0, 700.0], [33.5, 610.0], [33.5, 650.0], [34.0, 575.0], [34.0, 685.0], [34.5, 620.0], [35.0,
0, 700.0], [35.0, 725.0], [35.0, 720.0], [36.0, 714.0], [36.0, 850.0], [37.0, 1000.0], [38.5, 920.0], [38.5, 955.0], [39.5,
975.0], [41.0, 950.0], [8.4, 5.9], [13.7, 32.0], [15.0, 40.0], [16.2, 51.5], [17.4, 70.0], [18.0, 100.0], [18.7, 78.0], [19.
6, 85.0], [20.0, 85.0], [21.0, 110.0], [21.0, 115.0], [21.0, 125.0], [21.3, 130.0], [22.0, 120.0], [22.0, 120.0], [22.0, 130
5.0], [22.0, 110.0], [22.5, 130.0], [22.5, 150.0], [22.7, 145.0], [23.0, 150.0], [23.5, 170.0], [24.0, 225.0], [24.0, 145.0]
0], [25.0, 180.0], [25.6, 197.0], [26.5, 218.0], [27.3, 300.0], [27.5, 260.0], [27.5, 265.0], [27.5, 250.0], [28.0, 250.0],
[30.0, 320.0], [32.8, 514.0], [34.5, 556.0], [35.0, 840.0], [36.5, 685.0], [36.0, 700.0], [37.0, 700.0], [37.0, 690.0], [39.
9.0, 650.0], [39.0, 820.0], [40.0, 850.0], [40.0, 900.0], [40.0, 1015.0], [40.0, 820.0], [42.0, 1100.0], [43.0, 1000.0], [43
[43.5, 1000.0], [44.0, 1000.0], [30, 200], [31.7, 300], [32.7, 300], [34.8, 300], [35.5, 430], [36, 345], [40, 456], [40, 51
0], [42, 500], [43.2, 567], [44.8, 770], [48.3, 950], [52, 1250], [56, 1600], [56, 1550], [59, 1650]]
```

In [67]:
```
print(len(fish_data))
```

```
122
```

```
In [75]: # 정답 데이터를 만들기 위해 각 어종마다 class 번호를 부여한다.
         # smelt=0, bream=1, Perch=2, Pike=3
         fish_target = [0]*14 + [1]*35 + [2]*56 + [3]*17
         print(fish_target)
         print(len(fish_target))

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
122

In [76]: #사이킷런에서 제공되는 K-NN 알고리즘 사용 임포트
         from sklearn.neighbors import KNeighborsClassifier
         import sklearn
         kn = KNeighborsClassifier()

In [77]: #학습시키기
         kn.fit(fish_data, fish_target)

Out[77]: KNeighborsClassifier()

In [78]: #학습된 모델 점수
         kn.score(fish_data, fish_target)

Out[78]: 0.819672131147541
```

◆ 여러 class의 데이터가 섞여 있어도 각 class별로 별도 클래스를 부여하여 구분할 수 있다.

3 두 분류 모델의 예측 값 비교

◆ 첫번째 smelt와 bream을 가지고 학습한 모델을 사용한 예측

```
In [100]: #테스트 데이터를 가지고 결과를 예측해 보자.
          kn.predict([[21,150]])
Out[100]: array([0])
```

◆ 두번째 smelt, bream, perch, Pike를 가지고 학습한 모델을 사용한 예측

```
In [115]: #테스트 데이터를 가지고 결과를 예측해 보자.
          kn.predict([[21,150]])
Out[115]: array([2])
```

❖ 첫번째의 경우 검증을 별도로 진행하지 않더라도 결과가 잘못되었음을 직관적으로 알 수 있었다

❖ 두번째의 경우 3가지의 어종이 혼재되어 있어 얼마나 정확히 맞는지는 검증을 통하거나, 실제 산점도 등을 통해서 확인해야 만 알 수 있다.

→ 그러나 중요한 사실은 같은 알고리즘을 다른 데이터를 가지고 학습시켜 나온 모델은 서로 다른 예측을 하는 서로 다른 모델이 된 것이다. 즉, 머신러닝을 이용한 빅데이터 분석의 경우 데이터에 종속될 수 있다는 단점을 가지게 된다. 데이터의 특성이 모델의 성능을 좌우할 수 있다는 의미가 된다.

II-3. 데이터 전처리

1. 훈련 데이터 셋 vs. 테스트 데이터 셋
2. 머신러닝을 위한 데이터 전처리

1. 훈련 세트 vs. 테스트 세트

1 훈련 세트와 테스트 세트

- 훈련 세트: 훈련에 사용되는 데이터
- 테스트 세트: 평가에 사용하는 데이터

  → 주로 전체 데이터 세트에서 일부(20%~30%내외)를 테스트 세트로 분리하여 사용
- 파이썬 리스트에서 인덱스/ 슬라이싱 기능을 제공 => 이를 이용하여 데이터 분리
- 훈련 세트로 입력 값 중 0부터 34번째 인덱스까지 사용: train_input = fish_data[:35]
- 훈련 세트로 타겟 값 중 0부터 34번째 인덱스까지 사용: train_target = fish_target[:35]
- 테스트 세트로 입력 값 중 35부터 끝까지 사용: test_input = fish_data[35:]
- 테스트 세트로 타겟 값 중 35부터 끝까지 사용: test_target = fish_target[35:]
- 훈련 적용: kn = kn.fit(train_input, train_target) / kn.score(test_input, test_target)

2 샘플링 편향

- 훈련 세트와 테스트 세트에 샘플이 골고루 섞여 있지 않으면 샘플링이 한쪽으로 치우쳤다는 의미
- 해결하기 위해 데이터를 섞는다. → Numpy 이용

  np.random.seed(3)

  index = np.arrange(49)

  np.random.shuffle(index)

  → 넘파이는 리스트 대신 넘파이 배열을 인덱스로 전달할 수 있다.

1. 훈련 세트 vs. 테스트 세트

3 머신러닝을 위한 데이터 세트

- 머신러닝 모델의 효과성을 검증하기 위해 데이터를 다음과 같이 세 개로 나눔
  - Training Set(학습 세트)
  - Validation Set(검증 세트)
  - Test Set(평가 세트)
- 학습 세트와 검증 세트

  :학습데이터 세트를 사용하여 모델을 학습 시키고 이후 검증 세트를 통해 모델의 예측/분류 정확도를 계산

4 How to Split

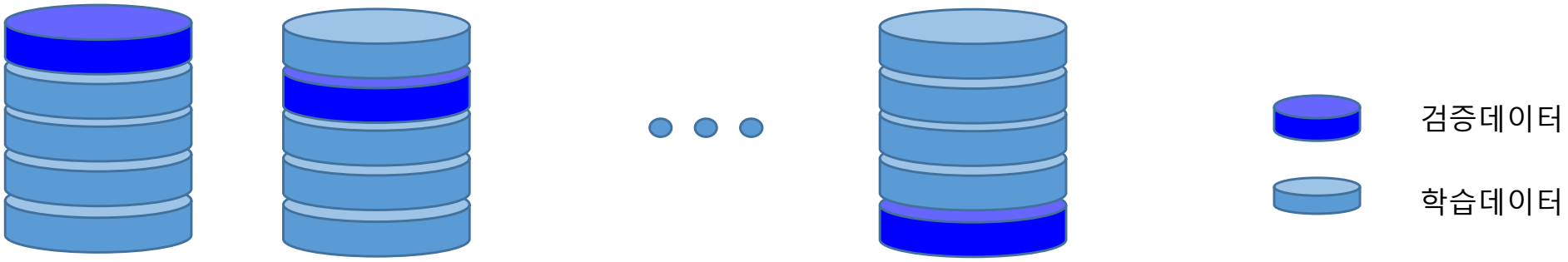
- 학습세트 vs 검증세트를 어떤 비율로 분할할지 판단하기 어려움.
- 학습 세트가 너무 작으면 알고리즘이 효과적으로 학습하기에 충분치 않을 수 있고, 검증 데이터가 너무 작으면 이를 통해 계산한 정확도(Accuracy), 정밀도(precision), 재현율(recall), 에 서로 차이가 많이 나서 신뢰가 어려움
- 전체 데이터 중 80%를 학습으로, 20%를 검증으로 사용하는 것이 좋음
- Scikit-learn에 제공

```
from sklearn.model_selection import train_test_split
training_data, validaion_data, training_labels, validation_labels = train_test_split(x, y, train_size=0.8, test_size=0.2)
```

1. 훈련 세트 vs. 테스트 세트

5 교차검증(N-Fold Cross-Validation

- 데이터가 충분하지 않을 경우 80:20으로 나누면 많은 양의 분산이 발생.
- N-Fold교차검증을 수행하는 것. 전체 프로세스를 N번 수행하고 모든 수행에서 나온 정확도의 평균을 구하는 방법
- 예를들어, 5번의 교차 검증이라고 하면 아래 그림과 같이 나타낼 수 있다.




```
In [36]: from sklearn.model_selection import KFold
```

4 모델 성능 개선 및 평가 세트

- 평가 데이터 셋은 검증세트와 비슷하지만, 모델을 구축하거나 튜닝할 때 포함된 적이 없는 데이터를 사용한다.

2. 머신러닝을 위한 데이터 전처리

1 훈련 데이터 셋과 테스트 데이터 셋 분류

- 훈련데이터와 테스트 데이터를 80% : 20% 정도의 비율로 나눈다.

2 샘플링 편향성 해결

- 훈련데이터와 테스트 데이터를 적절히 섞어서 사용

3 Feature Scaling

- Min-Max Normalization: 최솟값, 최대값을 이용해서 데이터의 크기를 0과 1 사이로 정규화
- 입력 변수의 크기를 조정해주어 일정 밤위 내에 속할 수 있도록 조절하는 기능

III-1.K-최근접 이웃 회귀

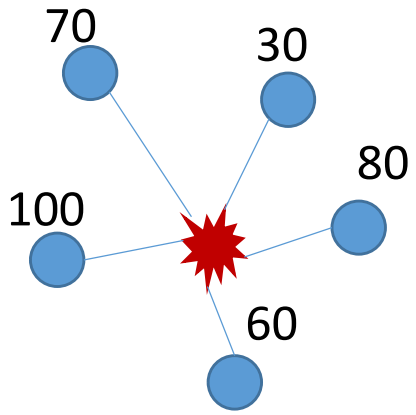
1. K-최근접 이웃 회귀

2. 결정계수

3. 과대적합 vs. 과소적합

1. K-최근접 이웃 회귀

- 회귀는 임의의 어떤 숫자를 예측하는 것
- 정해진 클래스가 없고 임의의 수치를 출력
- K-NN Regression(k-최근접 이웃 회귀)알고리즘은 주변의 가장 가까운 k개의 샘플을 통해 값을 예측하는 방식
- 동작원리



$$\text{✸} = \frac{100+80+70+60+30}{5}$$

$$= 68(\text{예측값})$$

2. 결정계수

- 사이킷런에서 k-최근접 이웃 회귀 알고리즘은 객체를 생성하고 fit()메서드로 회귀 모델을 훈련
- knr.score의 의미?

  : 결정계수, R^2

  : 회기를 평가하는 점수

  $R^2 = 1 - (\text{타겟}-\text{예측})^2\text{의 합} / (\text{타겟}-\text{평균})^2\text{의 합}$

  - 만약 타깃의 평균 정도를 예측하는 수준이라면(분자와 분모가 비슷해서) R^2 은 0에 가까워지고
  - 예측이 타깃에 아주 가까워지면 (분자가 0에 가까워지기 때문에) 1에 가까운 값이 된다.
  - 타깃과 예측값 사이의 차이를 구해 어느 정도 예측이 벗어났는지 가늠한다.
  - 사이킷런에서 제공하는 mean_absolute_error을 사용하여 절대값 오차를 평균하여 반환

In [1]:
```
import numpy as np
```

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

3. 과대적합 vs. 과소적합

◆ 과대적합: 훈련세트에서 점수가 높고, 테스트 세트에서 점수가 현저히 나쁠 때

: 모델이 훈련세트에 과대적합 되었다고 표현함

◆ 과소적합: 훈련세트에서 점수가 낮고, 테스트 세트에서 점수가 높을 때

또는 훈련세트나 테스트 세트 모두에서 점수가 낮을 때

모델이 훈련세트에 과소적합 되었다고 표현함

◆ 훈련세트와 테스트 세트의 점수를 비교하여 결정

◆ 주의! 데이터가 작으면 테스트 세트가 훈련 세트의 특징을 따르지 못할 수 있다.

◆ 해결: 훈련 세트보다 테스트 세트의 점수가 높으면 과소적합.

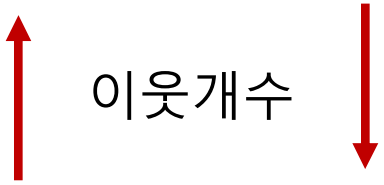
◆ 이를 해결하기 위한 방법은 모델을 조금 더 복잡하게 만든다.

→ 이웃의 개수를 줄이면, 훈련 세트에 있는 패턴에 민감해지고

이웃의 개수를 늘리면, 데이터 전반에 있는 일반적인 패턴을 따름.

◆ 훈련 세트보다 테스트 세트의 점수가 낮으면 과대적합.

◆ 이를 해결하기 위한 방법은 모델을 조금 덜 복잡하게 만든다.



III-2.Linear Regression

1. 선형회귀란?

2. 오차수정: 경사 하강법

3. 최적값과 학습률

1. 선형회귀

- 가장 직관적이고 간단한 모델은 **선(line)**이다. 그래서 데이터를 놓고 그걸 가장 잘 설명할 수 있는 선을 찾는 분석하는 방법을 **선형 회귀(Linear Regression)** 분석
- 독립변수 X를 사용해 종속 변수 Y의 값을 예측하는 것
- 하나의 독립변수를 사용하여 값을 예측하는 것을 단순 선형회귀
- 여러 개의 독립변수들을 사용하여 값을 예측하는 것을 다중 선형회귀
- 예) 학생 4명의 공부 시간을 표와 같이 나타날 때,

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

여기서 공부한 시간을 X, 성적을 Y라 하면,

X={2,4,6,8}

Y={81, 93, 91, 97}

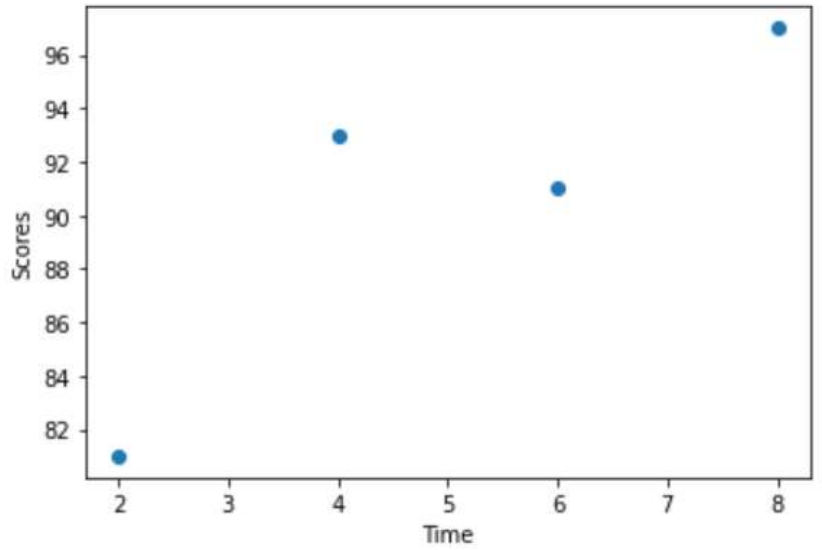
좌표 평면에 나타내면,

y = ax + b

선형회귀는 이 점들과 특징을 같이 해야 한다

즉, 기울기 a와 y절편 b를 구해야 한다.

```
In [15]: plt.scatter(X, Y)
         plt.xlabel('Time')
         plt.ylabel('Scores')
         plt.show()
```



◆ 기울기와 y절편을 구하는 방법

◆ 최소 제곱법

$$a = \frac{(x - x\text{평균})(y - y\text{평균})\text{의 합}}{(x - x\text{평균})^2\text{의 합}} \qquad b = y\text{의 평균} - (x\text{의 평균} \cdot \text{기울기 } a)$$

- 최소 제곱법으로 기울기를 구하면 2.3이 나오고 y 절편은 79가 나온다.

이제 최종적으로 식이 구해지는데, 다음과 같다.

$$Y = 2.3X + 79$$

위의 식을 이용해 예측한 값을 실제 종속값과 비교하면 다음 표와 같다.

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점
예측값	83.6점	88.2점	92.8점	97.4점

◆ 평균 제곱오차

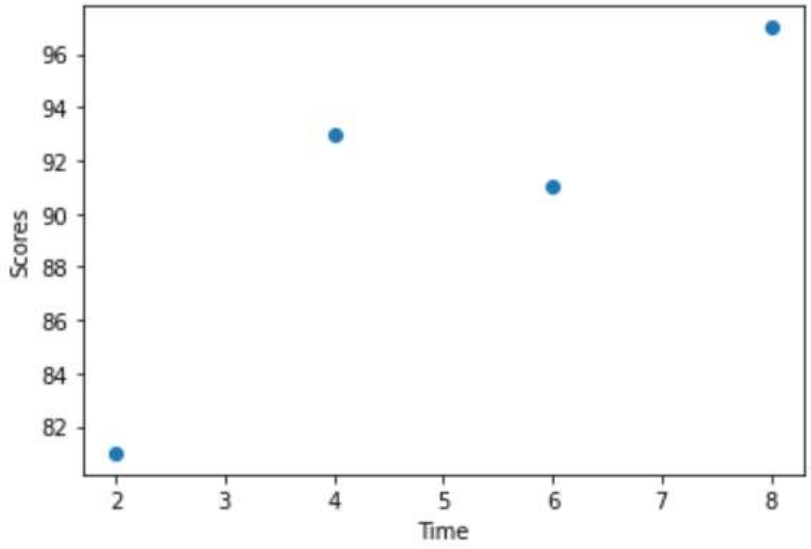
- 여러가지 변수가 있는 경우에 사용되는 방법.
- MSE(Mean Square Error, MSE: 평균 제곱 오차)
- 방법은 다르지만 결국 선형회귀의 목표는 실 데이터와 오차가 적은 선 그리기(기울기와 절편 구하기)
- 이 방법은 일단 임의의 선을 하나 긋고 조금씩 기울기와 y 절편을 수정해 나가는 방법

◆ 최소 제곱법

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [19]:
```
#시간과 성적 데이터 입력
X = [2, 4, 6, 8]
Y=[81, 93, 91, 97]
```

In [20]:
```
plt.scatter(X, Y)
plt.xlabel('Time')
plt.ylabel('Scores')
plt.show()
```



In [21]:
```
#평균을 구하는 메소드
meanx = np.mean(X)
meany = np.mean(Y)
```

◆ 최소 제곱법

In [25]:
```
#최소 제곱법에서 분모가 되는 값을 구하기
#X의 각 원소와 x의 평균 값들의 차를 구해 제곱한다.

diffx = sum([(i-meanx)**2 for i in X])
print(diffx)
```

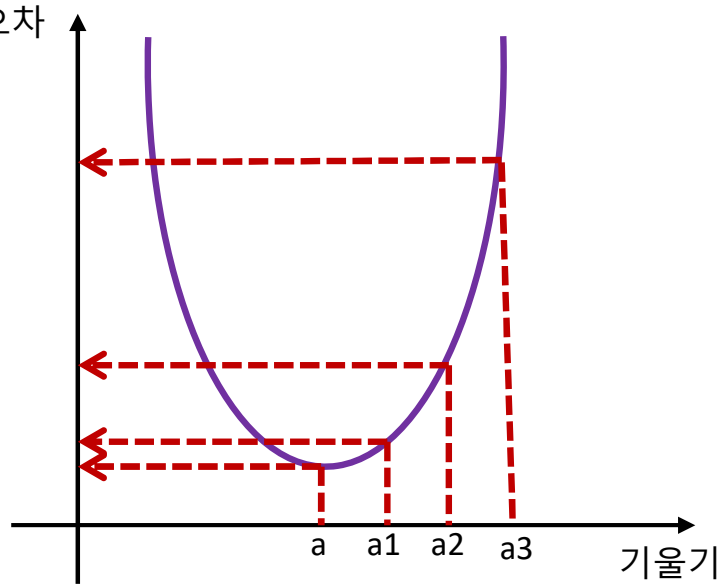
```
20.0
```

In [28]:
```
#최소 제곱법에서 분자가 되는 값을 구하기
#X의 각 원소와 X의 평균 값들의 차와 Y의 각 원소와 Y의 평균값들의 차를 구해 곱한 후

def top(X, meanx, Y, meany):

        diffvalue=0

        for i in range(len(X)):
            diffvalue += (X[i]-meanx) * (Y[i]-meany)
           # print(diffvalue)
        return diffvalue

diffnum = top(X,meanx, Y, meany)

print('분모:', diffx)
print('분자', diffnum)

a = diffnum / diffx
b = meany - (meanx*a)

print('기울기 a =', a)
print('y절편 b = ',b)
```

```
분모: 20.0
분자 46.0
기울기 a = 2.3
y절편 b =  79.0
```

2. 오차 수정: 경사 하강법

- 예측치와 실제 값의 차이를 오차라 하면, 오차를 줄일 수록 정확도가 높다고 할 수 있다.
- 예측은 정확도를 높일 수 있는 방향으로 학습해야 한다.
- 선형회귀에서 기울기 a를 너무 크게 잡거나 너무 작게 잡으면 오히려 오차가 커지는 것을 알 수 있다.
- 간단히 설명하면, 아래 그림과 같이 a값의 변화에 따른 오차를 그릴 수 있다.
- 오차가 가장 작은 지점은 기울기가 a일 때이다.
- 기울기의 시작점이 a1이라고 한다면, 머신이 a1에서 a쪽으로 갈 수 있도록 알려주어야 한다.
- 이렇게 오차를 비교하면서 작은 방향으로 이동 시키는 방법은 미분을 이용하는 **경사 하강법**이다.
- 즉 순간변화율인 그 점 에서의 미분값이 0일때가 된다.
- 편미분을 이용한다.



3. 최적값과 학습률

◆ 학습률(Learning Rate)

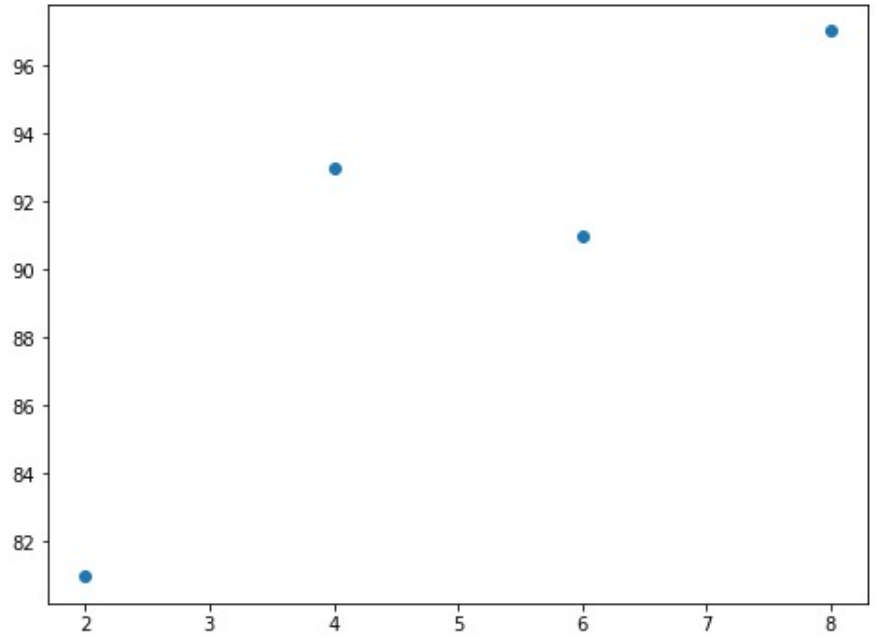
➢ 경사 하강법 등을 이용해서 기울기를 조절하면서 기울기의 적절한 값인 0에 도달할때까지 이동한 이동거리를 의미한다.

➢ 경사하강법은 오차에 따른 이차함수를 만들고 적절한 학습률을 설정해서 미분 값이 0인지점을 구하는 것이고, y절편인 b 역시 같은 방법으로 적정 값을 찾는다.

◆ 구현

➢ 다음은 경사 하강법을 구현한 예이다. 적절한 학습률과 시도회수를 설정하여 좀 더 빠르게 기울기와 절편을 구할 수 있도록 조절해 본다.

```
In [30]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt

In [32]: #data 입력
         Score_data = [[2,81], [4,93], [6,91], [8,97]]
         X = [i[0] for i in Score_data]
         Y = [i[1] for i in Score_data]

In [50]: # 그래프 시각화
         plt.figure(figsize=(6,4))
         plt.scatter(X, Y)
         plt.show()
```



3. 최적값과 학습률

In [39]:
```
#넘파이 배열로 변환
X_data = np.array(X)
Y_data = np.array(Y)
```

In [40]:
```
#가중치와 절편 초기화
a = 0
b = 0
```

In [41]:
```
#학습률과 학습 반복 횟수를 1000으로 설정 했을때
lr = 0.03
epochs = 1000
```

In [43]:
```
for i in range(epochs):
    #예측값과 오차계산
    predict_y = a * X_data + b
    err = Y_data -predict_y

    #편미분 연산
    a_dif = -(1/len(X_data)) * sum(X_data * err)
    b_dif = -(1/len(X_data)) * sum(err)

    #가중치와 절편 업데이트
    a = a - lr * a_dif
    b = b - lr * b_dif

    if i % 100 == 0:
        print('epoch=%.f,  기울기=%.04f   절편=%.04f' %(i, a, b))
```

```
print('epoch=%.f,  기울기=%.04f   절편=%.04f' %
epoch=0,  기울기=13.9200   절편=2.7150
epoch=100,  기울기=10.1307   절편=32.2700
epoch=200,  기울기=7.1088   절편=50.3032
epoch=300,  기울기=5.2531   절편=61.3774
epoch=400,  기울기=4.1135   절편=68.1780
epoch=500,  기울기=3.4136   절편=72.3542
epoch=600,  기울기=2.9839   절편=74.9189
epoch=700,  기울기=2.7200   절편=76.4938
epoch=800,  기울기=2.5579   절편=77.4609
epoch=900,  기울기=2.4584   절편=78.0549
```

3. 최적값과 학습률

In [41]:
```
#학습률과 학습 반복 횟수를 1000으로 설정 했을때
lr = 0.03
epochs = 1000
```

In [43]:
```
for i in range(epochs):
    #예측값과 오차계산
    predict_y = a * X_data + b
    err = Y_data -predict_y

    #편미분 연산
    a_dif = -(1/len(X_data)) * sum(X_data * err)
    b_dif = -(1/len(X_data)) * sum(err)

    #가중치와 절편 업데이트
    a = a - lr * a_dif
    b = b - lr * b_dif

    if i % 100 == 0:
        print('epoch=%.f,  기울기=%.04f   절편=%.04f' %(i, a, b))
```

```
epoch=0,  기울기=13.9200   절편=2.7150
epoch=100,  기울기=10.1307   절편=32.2700
epoch=200,  기울기=7.1088   절편=50.3032
epoch=300,  기울기=5.2531   절편=61.3774
epoch=400,  기울기=4.1135   절편=68.1780
epoch=500,  기울기=3.4136   절편=72.3542
epoch=600,  기울기=2.9839   절편=74.9189
epoch=700,  기울기=2.7200   절편=76.4938
epoch=800,  기울기=2.5579   절편=77.4609
epoch=900,  기울기=2.4584   절편=78.0549
```

In [44]:
```
#학습률과 학습 반복 횟수를 2000으로 설정 했을때
lr = 0.03
epochs = 2000
```

In [45]:
```
for i in range(epochs):
    #예측값과 오차계산
    predict_y = a * X_data + b
    err = Y_data -predict_y

    #편미분 연산
    a_dif = -(1/len(X_data)) * sum(X_data * err)
    b_dif = -(1/len(X_data)) * sum(err)

    #가중치와 절편 업데이트
    a = a - lr * a_dif
    b = b - lr * b_dif

    if i % 100 == 0:
        print('epoch=%.f,  기울기=%.04f   절편=%.04f' %(i, a, b))
```

```
epoch=0,  기울기=2.3973   절편=78.4196
epoch=100,  기울기=2.3597   절편=78.6436
epoch=200,  기울기=2.3367   절편=78.7811
epoch=300,  기울기=2.3225   절편=78.8656
epoch=400,  기울기=2.3138   절편=78.9175
epoch=500,  기울기=2.3085   절편=78.9493
epoch=600,  기울기=2.3052   절편=78.9689
epoch=700,  기울기=2.3032   절편=78.9809
epoch=800,  기울기=2.3020   절편=78.9883
epoch=900,  기울기=2.3012   절편=78.9928
epoch=1000,  기울기=2.3007   절편=78.9956
epoch=1100,  기울기=2.3005   절편=78.9973
epoch=1200,  기울기=2.3003   절편=78.9983
epoch=1300,  기울기=2.3002   절편=78.9990
epoch=1400,  기울기=2.3001   절편=78.9994
epoch=1500,  기울기=2.3001   절편=78.9996
epoch=1600,  기울기=2.3000   절편=78.9998
epoch=1700,  기울기=2.3000   절편=78.9999
epoch=1800,  기울기=2.3000   절편=78.9999
epoch=1900,  기울기=2.3000   절편=78.9999
```

3. 최적값과 학습률

In [46]:
```
#학습률과 학습 반복 횟수를 10000으로 설정 했을때
lr = 0.03
epochs = 10000
```

In [47]:
```
for i in range(epochs):
    #예측값과 오차계산
    predict_y = a * X_data + b
    err = Y_data -predict_y

    #편미분 연산
    a_dif = -(1/len(X_data)) * sum(X_data * err)
    b_dif = -(1/len(X_data)) * sum(err)

    #가중치와 절편 업데이트
    a = a - lr * a_dif
    b = b - lr * b_dif

    if i % 100 == 0:
        print('epoch=%.f,  기울기=%.04f   절편=%.04f' %(i, a, b))
```

```
epoch=0,  기울기=2.3000   절편=79.0000
epoch=100,  기울기=2.3000   절편=79.0000
epoch=200,  기울기=2.3000   절편=79.0000
epoch=300,  기울기=2.3000   절편=79.0000
epoch=400,  기울기=2.3000   절편=79.0000
epoch=500,  기울기=2.3000   절편=79.0000
epoch=600,  기울기=2.3000   절편=79.0000
epoch=700,  기울기=2.3000   절편=79.0000
epoch=800,  기울기=2.3000   절편=79.0000
epoch=900,  기울기=2.3000   절편=79.0000
epoch=1000,  기울기=2.3000   절편=79.0000
epoch=1100,  기울기=2.3000   절편=79.0000
epoch=1200,  기울기=2.3000   절편=79.0000
epoch=1300,  기울기=2.3000   절편=79.0000
epoch=1400,  기울기=2.3000   절편=79.0000
epoch=1500,  기울기=2.3000   절편=79.0000
epoch=1600,  기울기=2.3000   절편=79.0000
epoch=1700,  기울기=2.3000   절편=79.0000
epoch=1800,  기울기=2.3000   절편=79.0000
epoch=1900,  기울기=2.3000   절편=79.0000
epoch=2000,  기울기=2.3000   절편=79.0000
```

```
epoch=1700,  기울기=2.3000   절편=79.0000
epoch=1800,  기울기=2.3000   절편=79.0000
epoch=1900,  기울기=2.3000   절편=79.0000
epoch=2000,  기울기=2.3000   절편=79.0000
epoch=2100,  기울기=2.3000   절편=79.0000
epoch=2200,  기울기=2.3000   절편=79.0000
epoch=2300,  기울기=2.3000   절편=79.0000
epoch=2400,  기울기=2.3000   절편=79.0000
epoch=2500,  기울기=2.3000   절편=79.0000
epoch=2600,  기울기=2.3000   절편=79.0000
epoch=2700,  기울기=2.3000   절편=79.0000
epoch=2800,  기울기=2.3000   절편=79.0000
epoch=2900,  기울기=2.3000   절편=79.0000
epoch=3000,  기울기=2.3000   절편=79.0000
epoch=3100,  기울기=2.3000   절편=79.0000
epoch=3200,  기울기=2.3000   절편=79.0000
epoch=3300,  기울기=2.3000   절편=79.0000
epoch=3400,  기울기=2.3000   절편=79.0000
epoch=3500,  기울기=2.3000   절편=79.0000
epoch=3600,  기울기=2.3000   절편=79.0000
epoch=3700,  기울기=2.3000   절편=79.0000
epoch=3800,  기울기=2.3000   절편=79.0000
epoch=3900,  기울기=2.3000   절편=79.0000
epoch=4000,  기울기=2.3000   절편=79.0000
epoch=4100,  기울기=2.3000   절편=79.0000
epoch=4200,  기울기=2.3000   절편=79.0000
epoch=4300,  기울기=2.3000   절편=79.0000
epoch=4400,  기울기=2.3000   절편=79.0000
epoch=4500,  기울기=2.3000   절편=79.0000
epoch=4600,  기울기=2.3000   절편=79.0000
epoch=4700,  기울기=2.3000   절편=79.0000
epoch=4800,  기울기=2.3000   절편=79.0000
epoch=4900,  기울기=2.3000   절편=79.0000
epoch=5000,  기울기=2.3000   절편=79.0000
epoch=5100,  기울기=2.3000   절편=79.0000
epoch=5200,  기울기=2.3000   절편=79.0000
epoch=5300,  기울기=2.3000   절편=79.0000
epoch=5400,  기울기=2.3000   절편=79.0000
epoch=5500,  기울기=2.3000   절편=79.0000
epoch=5600,  기울기=2.3000   절편=79.0000
epoch=5700,  기울기=2.3000   절편=79.0000
epoch=5800,  기울기=2.3000   절편=79.0000
epoch=5900,  기울기=2.3000   절편=79.0000
epoch=6000,  기울기=2.3000   절편=79.0000
```

```
epoch=5900,  기울기=2.3000   절편=79.0000
epoch=6000,  기울기=2.3000   절편=79.0000
epoch=6100,  기울기=2.3000   절편=79.0000
epoch=6200,  기울기=2.3000   절편=79.0000
epoch=6300,  기울기=2.3000   절편=79.0000
epoch=6400,  기울기=2.3000   절편=79.0000
epoch=6500,  기울기=2.3000   절편=79.0000
epoch=6600,  기울기=2.3000   절편=79.0000
epoch=6700,  기울기=2.3000   절편=79.0000
epoch=6800,  기울기=2.3000   절편=79.0000
epoch=6900,  기울기=2.3000   절편=79.0000
epoch=7000,  기울기=2.3000   절편=79.0000
epoch=7100,  기울기=2.3000   절편=79.0000
epoch=7200,  기울기=2.3000   절편=79.0000
epoch=7300,  기울기=2.3000   절편=79.0000
epoch=7400,  기울기=2.3000   절편=79.0000
epoch=7500,  기울기=2.3000   절편=79.0000
epoch=7600,  기울기=2.3000   절편=79.0000
epoch=7700,  기울기=2.3000   절편=79.0000
epoch=7800,  기울기=2.3000   절편=79.0000
epoch=7900,  기울기=2.3000   절편=79.0000
epoch=8000,  기울기=2.3000   절편=79.0000
epoch=8100,  기울기=2.3000   절편=79.0000
epoch=8200,  기울기=2.3000   절편=79.0000
epoch=8300,  기울기=2.3000   절편=79.0000
epoch=8400,  기울기=2.3000   절편=79.0000
epoch=8500,  기울기=2.3000   절편=79.0000
epoch=8600,  기울기=2.3000   절편=79.0000
epoch=8700,  기울기=2.3000   절편=79.0000
epoch=8800,  기울기=2.3000   절편=79.0000
epoch=8900,  기울기=2.3000   절편=79.0000
epoch=9000,  기울기=2.3000   절편=79.0000
epoch=9100,  기울기=2.3000   절편=79.0000
epoch=9200,  기울기=2.3000   절편=79.0000
epoch=9300,  기울기=2.3000   절편=79.0000
epoch=9400,  기울기=2.3000   절편=79.0000
epoch=9500,  기울기=2.3000   절편=79.0000
epoch=9600,  기울기=2.3000   절편=79.0000
epoch=9700,  기울기=2.3000   절편=79.0000
epoch=9800,  기울기=2.3000   절편=79.0000
epoch=9900,  기울기=2.3000   절편=79.0000
```

3. 최적값과 학습률

In [48]:
```
#학습률을 0.05와 학습 반복 횟수를 1000으로 설정 했을때
lr = 0.05
epochs = 1000
```

In [49]:
```
for i in range(epochs):
    #예측값과 오차계산
    predict_y = a * X_data + b
    err = Y_data -predict_y

    #편미분 연산
    a_dif = -(1/len(X_data)) * sum(X_data * err)
    b_dif = -(1/len(X_data)) * sum(err)

    #가중치와 절편 업데이트
    a = a - lr * a_dif
    b = b - lr * b_dif

    if i % 100 == 0:
        print('epoch=%.f,  기울기=%.04f   절편=%.04f' %(i, a, b))
```

```
epoch=0,  기울기=2.3000   절편=79.0000
epoch=100,  기울기=2.3000   절편=79.0000
epoch=200,  기울기=2.3000   절편=79.0000
epoch=300,  기울기=2.3000   절편=79.0000
epoch=400,  기울기=2.3000   절편=79.0000
epoch=500,  기울기=2.3000   절편=79.0000
epoch=600,  기울기=2.3000   절편=79.0000
epoch=700,  기울기=2.3000   절편=79.0000
epoch=800,  기울기=2.3000   절편=79.0000
epoch=900,  기울기=2.3000   절편=79.0000
```

III-3. 다중 선형 회귀

1. 다중 선형 회귀 알고리즘
2. 예시1: 성적문제
3. 예시2: 주택임대료 예측하기

1. 다중 선형 회귀 알고리즘

- 입력 값으로 동작하는 독립변수가 2개 이상 들어올 때의 알고리즘
- 앞선 예의 시험 성적 데이터를 좀 더 활용해 보자. 두개의 독립변수가 존재하지만, 구하는 방식은 경사 하강법을 사용한다.

공부한 시간(X1)	2시간	4시간	6시간	8시간
과외 받는 횟수(X2)	0	4	2	3
성적(Y)	81점	93점	91점	97점

$$y = a1*x1 + a2*x2 + b$$

여기서 공부한 시간을 x1, 과외 받는 횟수를 x2성적을 Y라 하면,

X1 = [2, 4, 6, 8]

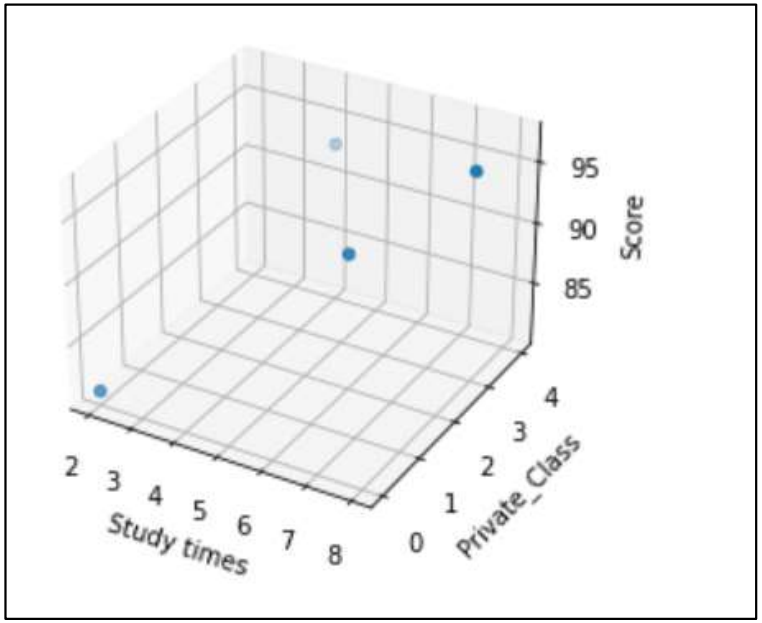
X2 = [0, 4, 2, 3]

Y =[81, 93, 91, 97]

그래프에 표시하면, 우측 그림과 같이 3D로 나타난다.

독립변수가 2개이더라도 구하는

방식은 똑같이 경사 하강법을 사용하면 된다.



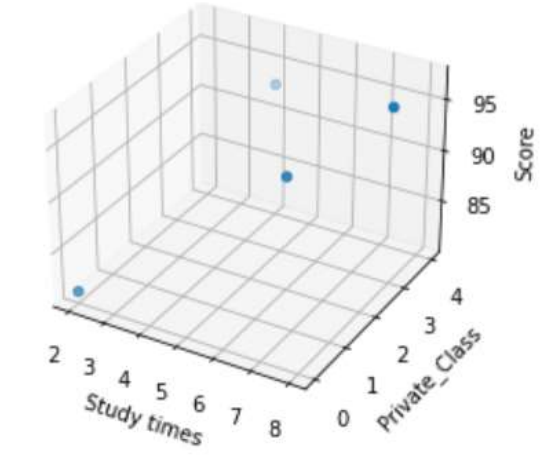
2. 예시1: 성적문제

- 구현(성적문제 예시)

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt

#3D 그래프를 그리기 위해 필요한 라이브러리
from mpl_toolkits import mplot3d
```

In [14]:
```
#독립 변수 2개와 1개의 종속변수에 해당하는 데이터를 입력한다.
input_data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]
X1 = [i[0] for i in input_data]
X2 = [i[1] for i in input_data]
Y = [i[2] for i in input_data]
```

In [15]:
```
ax = plt.axes(projection='3d')
ax.set_xlabel('Study times')
ax.set_ylabel('Private_Class')
ax.set_zlabel('Score')
ax.scatter(X1, X2, Y)
plt.show()
```



3. 다중 선형 회귀 알고리즘

- 구현(성적문제 예시)

In [16]:
```
#입력데이터를 머신러닝에서 사용하는 넘파이 어레이 형태로 바꿔준다.
X1_data = np.array(X1)
X2_data = np.array(X2)
Y_data = np.array(Y)
```

In [17]:
```
#기울기와 절편을 초기화 한다.
a1 = 0
a2 = 0
b = 0

#학습률은 0.03으로 반복횟수는 1000으로 맞춰준다
lr = 0.03
epochs = 1000
```

In [34]:
```
#Y값을 예측하고 실제 데이터와의 오차를 구한다.
for i in range(epochs) :
    predictY = a1 * X1_data + a2 * X2_data + b
    err = Y_data - predictY
    #편미분 연산
    #오차 함수를 a1로 미분
    diff_a1 = -(2/len(X1)) * sum(err * X1_data)
    diff_a2 = -(2/len(X2)) * sum(err * X2_data)
    diff_b = -(2/len(Y)) * sum(err)


    #학습률을 곱해 기존의 a1값 업데이트
    a1 -= lr * diff_a1
    a2 -= lr * diff_a2
    b -= lr * diff_b
    print(a1, a2, b)

    if i % 100 == 0:
        print('Epoch = %.04d, a1 = %.04f, a2 = %.04f, b = %.04f' %(i, a1, a2, b))
print('y = %.04fx1 + %.04fx2 + %.04f' %(a1, a2, b))
```

3. 다중 선형 회귀 알고리즘

- 구현(성적문제 예시)

In [34]:
```
#Y값을 예측하고 실제 데이터와의 오차를 구한다.
for i in range(epochs) :
    predictY = a1 * X1_data + a2 * X2_data + b
    err = Y_data - predictY
    #편미분 연산
    #오차 함수를 a1로 미분
    diff_a1 = -(2/len(X1)) * sum(err * X1_data)
    diff_a2 = -(2/len(X2)) * sum(err * X2_data)
    diff_b = -(2/len(Y)) * sum(err)


    #학습률을 곱해 기존의 a1값 업데이트
    a1 -= lr * diff_a1
    a2 -= lr * diff_a2
    b -= lr * diff_b
    print(a1, a2, b)

    if i % 100 == 0:
        print('Epoch = %.04d, a1 = %.04f, a2 = %.04f, b = %.04f' %(i, a1, a2, b))
print('y = %.04fx1 + %.04fx2 + %.04f' %(a1, a2, b))
```

```
-3.877252305911771e+205 -1.7590261602377979e+205 -6.531614918643058e+204
4.669790697274192e+205 2.1185838194785067e+205 7.866724210531866e+204
-5.624329663328543e+205 -2.5516376627106934e+205 -9.474739490218605e+204
6.773983292283734 5e+205 3.073210841082673e+205 1.1411444713839578e+205
-8.15863443128666e+205 -3.7013973464064362e+205 -1.374402648130361e+205
9.826318269665454e+205 4.457989713181556e+205 1.6553404819083275e+205
-1.1834888736586474e+206 -5.3692350274491e+205 -1.9937040391853177e+205
1.4254025522435067e+206 6.466745469318725e+205 2.4012315528473715e+205
-1.716765134987087e+206 -7.788594976968668e+205 -2.8920606354120182e+205
2.067684335255586e+206 9.380640076692603e+205 3.483218729564735e+205
-2.4903339560734347e+206 -1.1298110700153491e+206 -4.195213810329409e+205
2.999376213780602e+206 1.3607526176180539e+206 5.0527458310313805e+205
-3.612470387697504e+206 -1.6389002865135924e+206 -6.085563593956681e+205
4.350885441457307e+206 1.9739033490422883e+206 7.3294967715665825e+205
-5.240237868565827e+206 -2.377383458543955e+206 -8.82769888030642e+205
6.311380358926139e+206 2.863337818288147e+206 1.0632144327243787e+206
-7.60147211522298e+206 -3.448624761047376e+206 -1.2805431463858253e+206
9.155267949711086e+206 4.153548584644243e+206 1.5422954197056143e+206
-1.1026670881703063e+207 -5.002563932110909e+206 -1.8575517493170214e+206
```

Python 기반 AI를 활용한 빅데이터 분석가 양성 과정

Q&A / 감사합니다.

메디치교육센터
서울특별시 금천구 가산디지털1로 168(우림라이온스밸리 A동 304 ~ 307호)
Tel 02-861-8568 · **Fax** 070-7525-8538