파이썬 기초~응용

『4과목 :』
데이터 분석과 인사이트 도출

2025.06.23-07.04(10일, 70시간)

Prepared by DaeKyeong K
Ph.D.

SAMSUNG DISPLAY

KOREATECH 한국기술교육대학교

목차

『 4과목 :』
데이터 분석과 인사이트 도출

- 통계와 확률 / 통계분석
- 머신러닝 기반 데이터 분석-지도 : 분류 및 회귀
- 머신러닝 기반 데이터 분석-비지도
- 기타 데이터 마이닝

- 『4과목』 Self 점검

Syllabus

학습목표

- 이 워크샵에서는 머신러닝 기반 데이터 분석과 지도-비자율학습 분류에 대해 확인할 수 있습니다.
- 이 워크샵에서는 지도-비자율학습 모델, K-최근접 이웃(K-Nearest Neighbor) 기법, 회귀, 나이브 베이즈(Naive Bayes) 기법에 대해 알 수 있습니다.

눈높이 체크

- 지도-비자율학습 분류를 알고 계신가요?
- K-최근접 이웃(K-Nearest Neighbor) 을 알고 계시나요?
- 회귀
- 나이브 베이즈(Naive Bayes) 기법

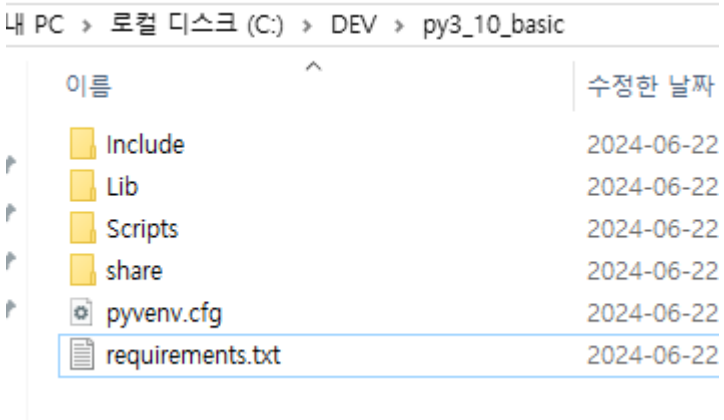
먼저, pip이용하여 requirements.txt 만들기

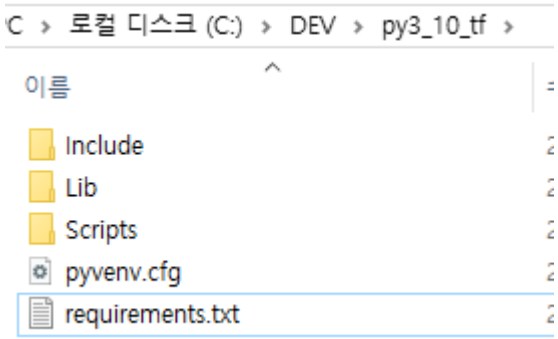
```
(py3_10_basic) c:\DEV\envs\py3_10_basic>pip freeze > requirements.txt

(py3_10_basic) c:\DEV\envs\py3_10_basic>dir/w/p
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: BED0-C858

 c:\DEV\envs\py3_10_basic 디렉터리

[.]             [..]            [Include]       [Lib]           pyvenv.cfg      requirements.txt
[Scripts]       [share]
               2개 파일               1,596 바이트
               6개 디렉터리  40,564,559,872 바이트 남음

(py3_10_basic) c:\DEV\envs\py3_10_basic>
```

TensorFlow 2 가상환경 만들기

```
c:\DEV>cd envs

c:\DEV\envs>python -m venv py3_10_tf

c:\DEV\envs>cd py3_10_tf

c:\DEV\envs\py3_10_tf>Scripts\activate

(py3_10_tf) c:\DEV\envs\py3_10_tf>dir/w/p
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: BED0-C858

 c:\DEV\envs\py3_10_tf 디렉터리

[.]          [..]         [Include]    [Lib]        pyvenv.cfg    [Scripts]
               1개 파일                 118 바이트
               5개 디렉터리  56,743,809,024 바이트 남음

(py3_10_tf) c:\DEV\envs\py3_10_tf>
```

requirements.txt 복사






```
(py3_10_tf) c:\DEV\py3_10_tf>dir/w/p
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: BED0-C858

 c:\DEV\py3_10_tf 디렉터리

[.]             [..]            [Include]       [Lib]           pyvenv.cfg      requirements.txt
[Scripts]
               2개 파일                 953 바이트
               5개 디렉터리  68,334,325,760 바이트 남음

(py3_10_tf) c:\DEV\py3_10_tf>
```

다음, requiremnets.txt 이용 패키지 설치

```
(py3_10_tf) c:\DEV\py3_10_tf>pip install -r requirements.txt
Collecting asttokens==2.4.1
  Using cached asttokens-2.4.1-py2.py3-none-any.whl (27 kB)
Collecting colorama==0.4.6
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting comm==0.2.2
  Using cached comm-0.2.2-py3-none-any.whl (7.2 kB)
Collecting contourpy==1.2.1
  Downloading contourpy-1.2.1-cp310-cp310-win_amd64.whl (187 kB)
     ---------------------------------------- 187.5/187.5 kB 5.7 MB/s eta 0:00:00
Collecting cycler==0.12.1
  Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
```

쥬피터 노트북 내보내기

```
(py3_10_tf) c:\DEV\py3_10_tf> pip install ipykernel
(py3_10_tf) c:\DEV\py3_10_tf> python.exe -m pip install --upgrade pip

(py3_10_tf) c:\DEV\py3_10_tf>python -m ipykernel install --user --name py3_10_tf --display-name "py3_10_tf"
Installed kernelspec py3_10_tf in C:\Users\k8s\AppData\Roaming\jupyter\kernels\py3_10_tf

(py3_10_tf) c:\DEV\envs\py3_10_tf>deactivate
c:\DEV\envs\py3_10_tf>cd ..

c:\DEV\envs>cd ..

c:\DEV>jupyter notebook
```

설치 된 라이브러리 확인

```
import sys
print("python 버전 : {}".format(sys.version))
import numpy as np
print("numpy 버전 : {}".format(np.__version__))
import pandas as pd
print("pandas 버전 : {}".format(pd.__version__))
import matplotlib
print("matplotlib 버전 : {}".format(matplotlib.__version__))
import scipy as sp
print("scipy 버전 : {}".format(sp.__version__))
import IPython
print("IPython 버전 : {}".format(IPython.__version__))
import sklearn
print("sklearn : {}".format(sklearn.__version__))
```

```
python 버전 : 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)]
numpy 버전 : 2.0.0
pandas 버전 : 2.2.2
matplotlib 버전 : 3.9.0
scipy 버전 : 1.13.1
IPython 버전 : 8.25.0
sklearn : 1.5.0
```

TensorFlow 2 설치

```
(py3_10_tf) c:\DEV\py3_10_tf>python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'tensorflow'
>>> quit()

(py3_10_tf) c:\DEV\py3_10_tf>pip install --upgrade tensorflow
Collecting tensorflow
  Downloading tensorflow-2.16.1-cp310-cp310-win_amd64.whl.metad
  ...

    Attempting uninstall: numpy
    Found existing installation: numpy 2.0.0
    Uninstalling numpy-2.0.0:
      Successfully uninstalled numpy-2.0.0
Successfully installed MarkupSafe-2.1.5 absl-py-2.1.0 astunparse-1.6.3 certifi-2024.6.2 charset-normalizer-3.3.2 flatbuffers-24.3.25
gast-0.5.4 google-pasta-0.2.0 grpcio-1.64.1 h5py-3.11.0 idna-3.7 keras-3.3.3 libclang-18.1.1 markdown-3.6 markdown-it-py-3.0.0
mdurl-0.1.2 ml-dtypes-0.3.2 namex-0.0.8 numpy-1.26.4 opt-einsum-3.3.0 optree-0.11.0 protobuf-4.25.3 requests-2.32.3 rich-13.7.1
tensorboard-2.16.2 tensorboard-data-server-0.7.2 tensorflow-2.16.1 tensorflow-intel-2.16.1 tensorflow-io-gcs-filesystem-0.31.0
termcolor-2.4.0 urllib3-2.2.2 werkzeug-3.0.3 wheel-0.43.0 wrapt-1.16.0

(py3_10_tf) c:\DEV\py3_10_tf>
```

GPU에서 TensorFlow 2 설치

https://www.tensorflow.org/install/pip?hl=ko#windows-native

★ **참고:** Conda와 함께 TensorFlow를 설치하지 마십시오. 최신 안정 버전이 없을 수도 있습니다. TensorFlow는 PyPI에만 공식적으로 출시되므로 pip를 권장합니다.

```
# Anything above 2.10 is not supported on the GPU on Windows Native
pip install "tensorflow<2.11"
```

pip install "numpy<2"

(py3_10_tf_gpu) c:\DEV\envs\py3_10_tf_gpu>pip install "tensorflow<2.11"

GPU에서 TensorFlow 2 설치

```
C:\DEV\envs\py3_10_tf_gpu>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:41:10_Pacific_Daylight_Time_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

GPU에서 TensorFlow 2 설치

TensorFlow가 GPU

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
print("GPU available:", tf.test.is_gpu_available())
```

```
import tensorflow as tf

print(tf.config.list_physical_devices('GPU'))
```

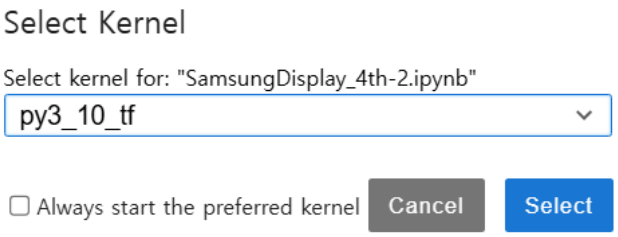
. is_build_with_cuda()
import tensorflow as tf print(tf.test.is_built_with_cuda())
output : True

. gpu_device_name()
tf.test.gpu_device_name()
output : '/device:GPU:0'

TensorFlow 2 설치 확인

```
(py3_10_tf) c:\DEV\envs\py3_10_tf>deactivate
c:\DEV\envs\py3_10_tf>cd ..
c:\DEV\envs>cd ..
c:\DEV>
c:\DEV>jupyter notebook
```

```
import tensorflow as tf
with tf.compat.v1.Session() as sess:
   helloworld = tf.constant("Hello World!")
   print(sess.run(helloworld))
++++++++++++++++++++++++++++++++++++++++
helloworld = tf.constant("Hello World!")
tf.print(helloworld)
```




```
[1]: import tensorflow as tf
```

```
[2]: with tf.compat.v1.Session() as sess:
        helloworld = tf.constant("Hello World!")
        print(sess.run(helloworld))
```

```
WARNING:tensorflow:From C:\Users\k8s\AppData\Local\Temp\ipykernel_9808\1734762880.py:1: The name tf.Session is deprecated. Please use tf.compa
t.v1.Session instead.

b'Hello World!'
```

```
[3]: helloworld = tf.constant("Hello World!")
     tf.print(helloworld)
```

```
Hello World!
```

1. 머신러닝(Machine Learning)

머신러닝이란?

- 학습이란? <표준국어대사전>

"경험의 결과로 나타나는, 비교적 지속적인 행동의 변화나 그 잠재력의 변화. 또는 지식을 습득하는 과정[국립국어원2017]"

- 기계 학습이란?
- 인공지능 초창기 사무엘의 정의
  - 1959년 Arthur Samuel이 체스 자동 대국 프로그램을 제작하며 기계가 기계의 설계자에게 가르침을 받으며 종국에는 설계자의 실력보다 나아지는 것을 ' 머신러닝'이라 부르며 머신러닝의 개념이 도입되게 되었다.

"Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort. 컴퓨터가 경험을 통해 학습할 수 있도록 프로그래밍할 수 있다면, 세세하게 프로그래밍해야 하는 번거로움에서 벗어날 수 있다[Samuel1959]."

1. 머신러닝(Machine Learning)

머신러닝이란?

- **현대적 정의**

"A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.
어떤 컴퓨터 프로그램이 *T*라는 작업을 수행한다. 이 프로그램의 성능을 *P*라는 척도로 평가했을 때 경험 *E*를 통해 성능이 개선된다면 이 프로그램은 학습을 한다고 말할 수 있다[Mitchell1997(2쪽)]."

"Programming computers to optimize a performance criterion using example data or past experience
사례 데이터, 즉 과거 경험을 이용하여 성능 기준을 최적화하도록 프로그래밍하는 작업[Alpaydin2010]"

"Computational methods using experience to improve performance or to make accurate predictions
성능을 개선하거나 정확하게 예측하기 위해 경험을 이용하는 계산학 방법들[Mohri2012]"

1. 머신러닝(Machine Learning)

머신러닝이란?

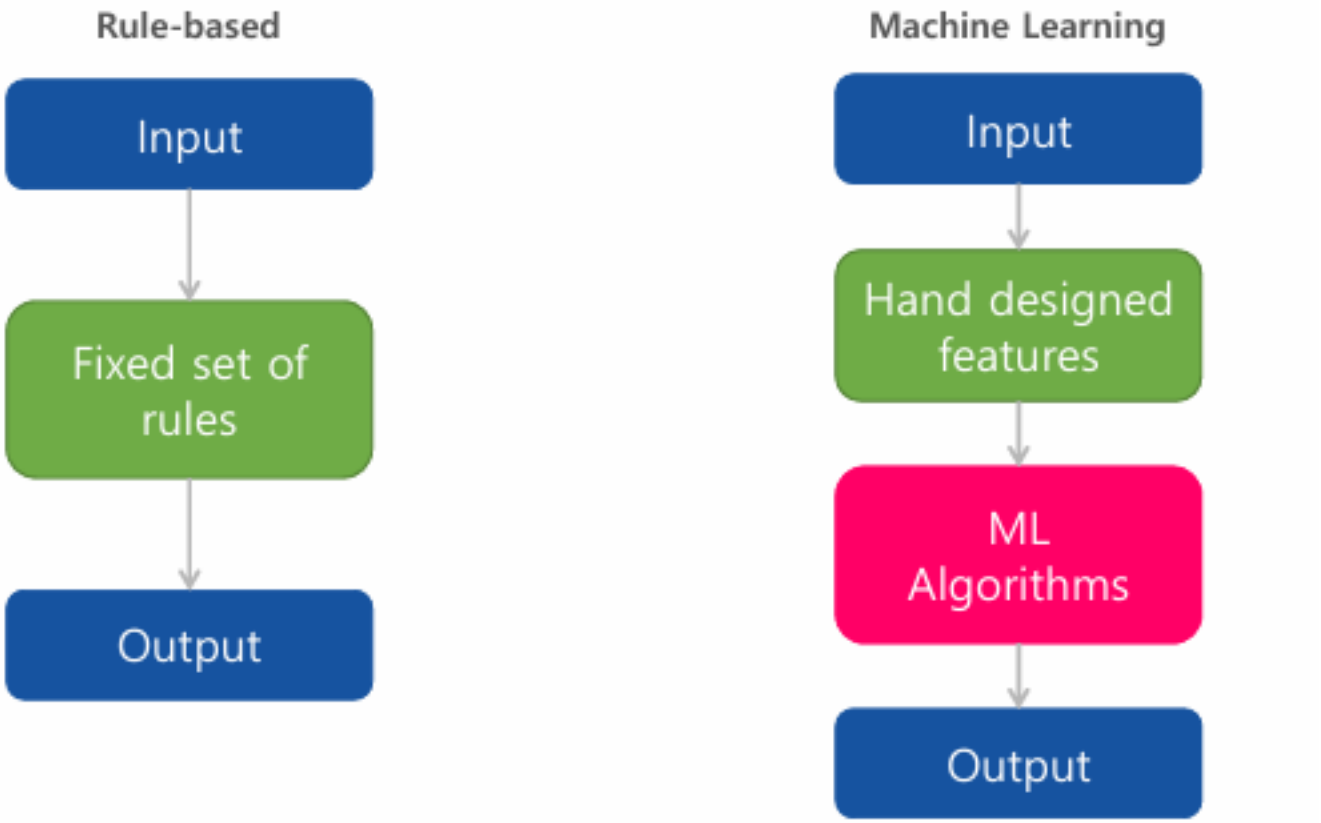
- 머신러닝을 위해서는 수행하고자 하는 '작업'이 있어야 하며, 작업을 수행하기 위해 주 어진 데이터로부터 학습한 '모델', 그리고 작업수행을 위해 모델을 학습하는 과정에서 개체를 측정한 '특성(Features)'이 필요하다.

주요 제안자 및 출처	머신러닝에 대한 다양한 정의
Arthur Samuel (1959)	컴퓨터에게서 배울 수 있는 능력, 즉 코드로 정의하지 않은 동작을 실행하는 능력에 관한 연구 분야
Tom Mitchell (1996, 1997)	작업 T에 대한 성능을 P로 측정할 수 있고, 성능 P가 경험 E로 개선된다면 프로그램은 E,T,P 로부터 배웠다고 할 수 있다.
Peter Flach (2012)	정확한 '작업(Task)'을 성취할 수 있는 올바른 '모델(Model)'을 구축하기 위해 올바른 '특성(Features)'을 활용하는 것
Jason Bell (2015)	컴퓨터 프로그램이 어떤 것을 학습한 후에 최초 학습에 들인 시간과 노력보다 더 빠르고 수월하게 배운 것을 해낼 수 있게 하는 것
Wekipedia	인공지능의 한 분야로 컴퓨터가 학습할 수 있도록 하는 알고리즘과 기술을 개발하는 분야

1. 머신러닝(Machine Learning)

왜 머신러닝을사용하는가

- 머신러닝 모델을 통해 코드를 간단하고 더 잘 수행할 수 있도록 함
- 머신러닝 기법으로 복잡한 문제를 해결할 수 있음
- 새로운 데이터에 대해 유동적으로 적응할 수 있음



1. 머신러닝(Machine Learning)

최근 환경

- 도구가 다양하고 체계화되어 환경에 적합한 제품을 선택하여 활용 가능하다.
- 알고리즘에 대한 깊은 이해가 없어도 분석에 큰 어려움이 없다.
- 분석 결과의 품질은 분석가의 경험과 역량에 따라 차이가 나기 때문에 분석 과제의 복잡성이나 중요도가 높으면 풍부한 경험을 가진 전문가에게 의뢰할 필요가 있다.
- 국내에서 머신러닝이 적용된 시기는 1990년대 중반이다.
- 2000년대에 비즈니스 관점에서 머신러닝이 CRM의 중요한 요소로 부각되었다.대중화를 위해 많은 시도가 있었으나 통계학 전문가는 대기업 위주로 진행되었다.

1. 머신러닝(Machine Learning)

머신러닝 추진단계

- [1단계] 목적 설정
  - 머신러닝을 통해 무엇을 왜 하는지 명확한 목적을 설정. 전문가가 참여해 목적에 따라 사용할 모델과 필요할 데이터를 정의
- [2단계] 데이터 준비
  - 고객정보, 거래정보, 상품정보, 마스터 정보, 웹로그 데이터, 소셜 네트워크 데이터 등 다양한 데이터를 활용
  - IT부서와 사전에 협의하고 일정을 조율하여 데이터 접근 부하에 유의하여야 하며, 필요시 다른 서버에 저장하여 운영에 지장이 없도록 데이터 준비
  - 데이터 정재를 통해 데이터 품질을 보장하고, 필요시 데이터 보강하여 충분한 데이터를 확보
- [3단계] 가공
  - 모델링 목적에 따라 목적 변수 정의. 필요한 데이터를 머신러닝 소프트웨어에 적용할 수 있는 형식으로 가공
- [4단계] 기법 적용
  - 1단계에서 명확한 목적에 맞게 머신러닝 기법을 적용하여 정보 추출
- [5단계] 검증
  - 머신러닝으로 추출된 정보를 검증. 테스트 데이터와 과거 데이터를 활용하여 최적의 모델 선정
  - 검증이 완료되면 IT 부서와 협의해 상시 데이터마이닝 결과를 업무에 적용하고 보고서를 작성하여 추가 수익과 투자 대비 성과(ROI)등으로 기대효과를 전파

1. 머신러닝(Machine Learning)

데이터 전처리

- 값이 누락된 데이터 정리
  - 데이터가 충분한 경우
    - 값이 누락된 데이터를 제거함
  - 데이터가 제한적일 경우
    - 누락된 값을 추정해서 채움

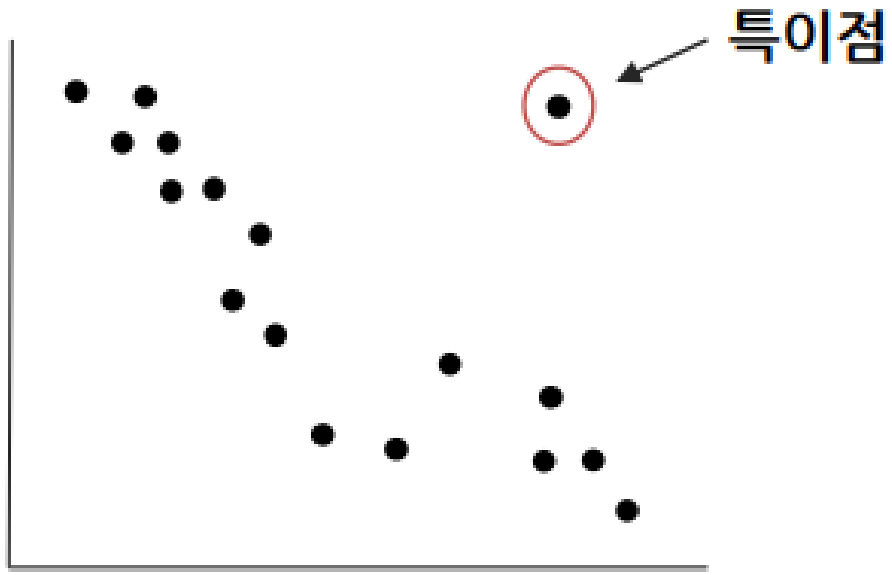
이름	나이	키	몸무게
김철수	29	175cm	80kg
김영희	27	160cm	50kg
홍길동	25		60kg

<누락된 값의 예>

1. 머신러닝(Machine Learning)

데이터 전처리

- **특이점 제거**
  - 특이점은 머신러닝 결과에 영향을 미칠 수 있으므로 제거함
  - 신뢰 구간을 정하고 학습 데이터 중 범위를 벗어난 값을 제거함
  - 신뢰 구간을 벗어나는 데이터를 처리하지 않으므로 불필요한 머신러닝 수행시간이 단축됨



〈특이점 제거의 예〉

1. 머신러닝(Machine Learning)

데이터 전처리

- **데이터 변환**
  - 수집된 데이터를 머신러닝에 적합한 값으로 변환함
  - 데이터 변환을 통해 데이터 처리를 위한 연산시간을 축소할 수 있음
  - 데이터 변환 기술 : 표준화, 정규화, 이산화 등

표준화	• 데이터가 가우시안 분포를 따르고, 평균의 0, 편차는 1이라는 가정을 따름 $X = \frac{X - mean(X)}{st.dev}$
정규화	• 데이터의 범위를 0과 1 사이로 한정함 $X = \frac{X - min}{max - min}$

이산화	• 의사 결정 트리, 나이브 베이즈 기법 등에서 이산화된 값을 사용하는 것이 유리함 • 구간 선택 방법	
	동등 폭	구간을 동등한 폭으로 나눔
	동등 빈도	각 구간에 동등한 개수의 데이터가 존재하도록 나눔
	민 엔트로피	데이터의 무질서도가 가장 낮은 수준까지 나눔

1. 머신러닝(Machine Learning)

데이터 전처리

- 데이터 축소
  - 데이터의 속성 중 예측력이 떨어지는 속성은 전체 모델에 기여하는 바가 적을 뿐 아니라 신뢰성을 떨어뜨리는 요소가 됨
  - 이 문제를 해결하기 위한 방법
    - 예측력이 떨어지는 속성 자체를 제거함
    - 고차원의 데이터를 저차원으로 변환함

  - 너무 많은 데이터가 존재하는 경우 데이터가 중복되거나 불필요한 반복이 발생하는 문제가 있음
  - 이 문제를 해결하기 위한 방법
    - 원본 데이터의 분포를 그대로 유지하는 하위 집합을 선택 할 수 있음

데이터 학습 및 평가

- **데이터 학습 및 평가**
  - **학습 데이터와 검증 데이터의 구분**
    - **모델이 완성된 뒤에 모델을 평가하기 위해 데이터를 학습 데이터와**
    - **검증 데이터 두 가지로 나눠야 함**
    - **학습 데이터와 검증 데이터의 비율은 70:30 정도가 적당**

  - **교차 평가**
    - **데이터가 충분하지 못한 경우에는 교차 평가를 시행함**

1. 머신러닝(Machine Learning)

Machine Learning 학습

- 지도학습(supervised learning): AI가 레이블된 데이터를 통해 학습하는 방법으로, AI 응용 분야에서 가장 널리 사용됨
  - 예: 이미지 분류 작업에서 각 이미지에 “고양이”, “개” 등의 레이블을 붙여 학습

- 비지도학습(unsupervised learning): AI가 레이블이 없는 데이터를 통해 학습하는 방법 으로, 복잡하고 대량의 데이터에서 숨겨진 패턴·구조 등을 찾는데 유용함
  - 예: 클러스터링(clustering) 방법을 사용하여 비슷한 특성을 가진 데이터를 그룹화

- 강화학습(reinforcement learning): AI가 주어진 환경에서 행동을 선택하고 그 결과로서 ‘보상’이나 ‘처벌’을 받도록 하여 AI가 더 나은 행동을 선택하도록 유도
  - 예: 로봇학습, 자율주행차 등 다양한 분야에 적용

1. 머신러닝(Machine Learning)

Machine Learning 학습

지도학습 Supervised Learning

- 입력과 결과가 레이블로 표시
- 입력과 출력에 매핑되는 일반적인 규칙을 학습

비지도학습 Unsupervised Learning

- 원하는 출력 없이 입력 데이터 사용
- 입력 데이터의 구조나 패턴을 찾는 것이 목표

준지도학습 Semi-supervised Learning

- 레이블이 있는 것과 없는 것이 혼합된 경우 사용
- 일반적으로 일부 데이터에만 레이블이 있음

강화학습 Reinforcement Learning

- 동적 환경과 함께 상호작용
- 어떤 지도가 없이 일정한 목표를 수행

1. 머신러닝(Machine Learning)

지도 학습

- 데이터에서 하나의 함수를 유추해 내기 위한 방법
- 입력 데이터에 결과 값이 포함되어 있음
- 알고리즘에 주입하는 훈련 데이터에 레이블(label) 이라는 원하는 답이 포함된다.
  - 분류
  - 특성(예측 변수)
  - 회귀
- 중요한 지도학습 알고리즘들
  - k-최근접 이웃
  - 선형 회귀
  - 로지스틱 회귀
  - 서포트 벡터 머신
  - 결정 트리와 랜덤 포레스트
  - 신경망

입력	정답
고양이 사진	고양이
고양이 사진	고양이
강아지 사진	강아지
강아지 사진	강아지

학습 →

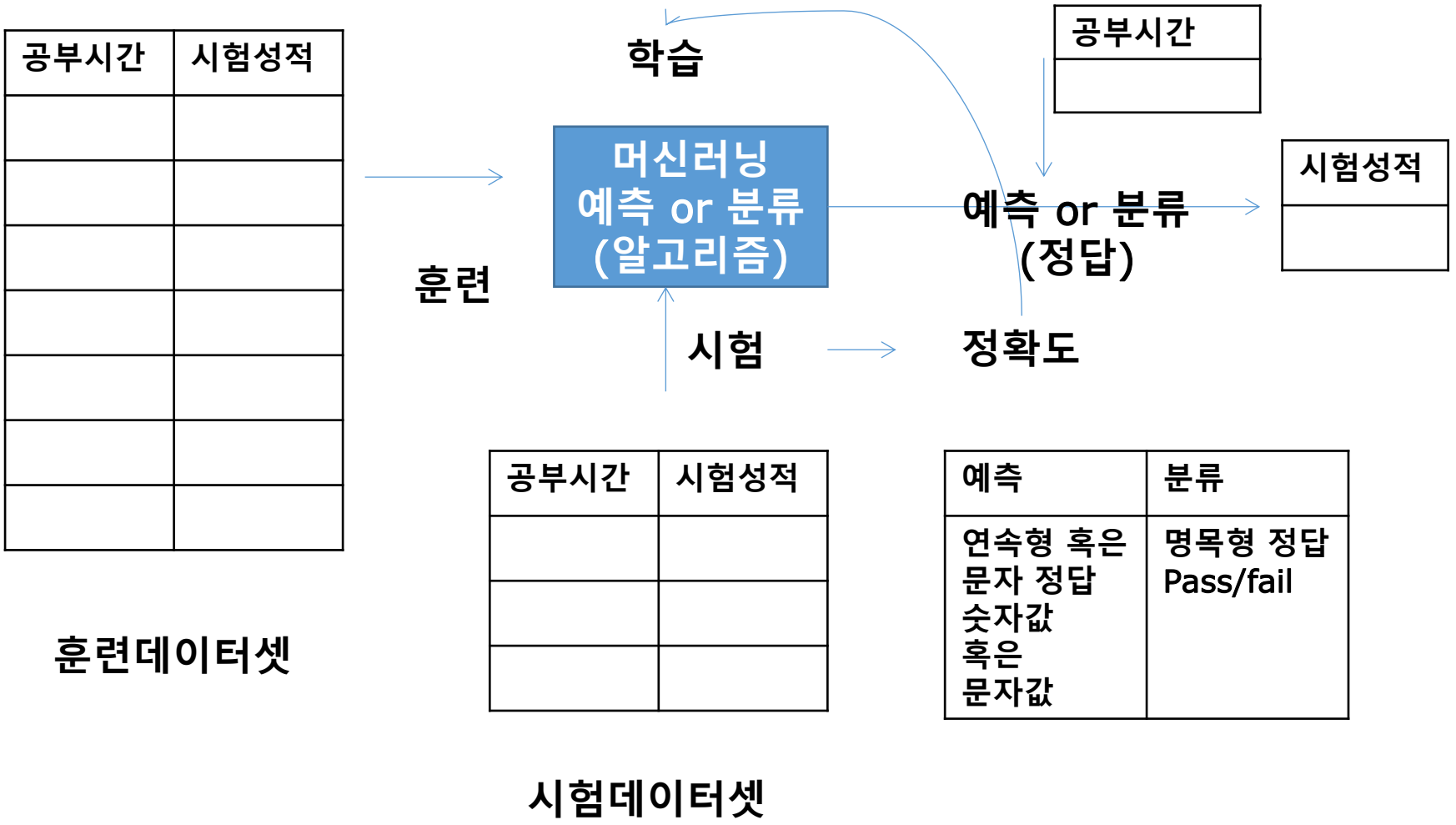
사진에서 고양이와 강아지를 구분해 내는 모델

〈지도학습의 예〉

1. 머신러닝(Machine Learning)

지도 학습

- 지도학습 개념도 : 입력에 대한 정답을 찾는 과정



1. 머신러닝(Machine Learning)

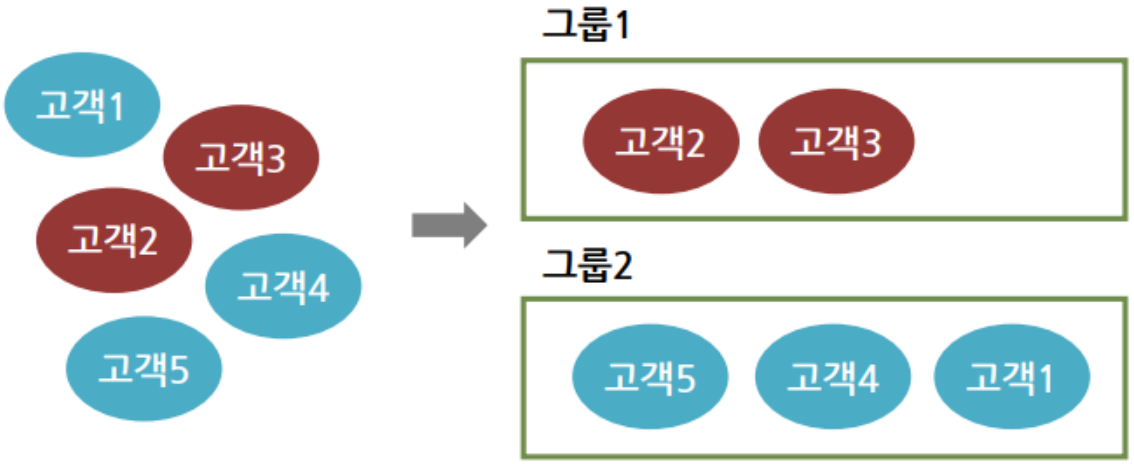
비지도 학습

- 데이터가 어떻게 구성되어 있는지 알아내는 방법
- 입력 데이터에 결과 값이 포함되어 있지 않음
- 훈련 데이터에 레이블이 없어서, 시스템이 아무런 도움 없이 학습해야 한다.
- 중요한 비지도학습 알고리즘들

◦ 군집
– k-평균
– DBSCAN
– 계층 군집 분석
– 이상치 탐지와 특이치 탐지
– 원-클래스 SVM
– 아이솔레이션 포레스트

◦ 시각화와 차원 축소
– 주성분 분석(PCA)
– 커널 PCA
– 지역적 선형 임베딩
– t-SNE

◦ 연관 규칙 학습
– 어프라이어리(Apriori)
– 이클렛(Eclat)

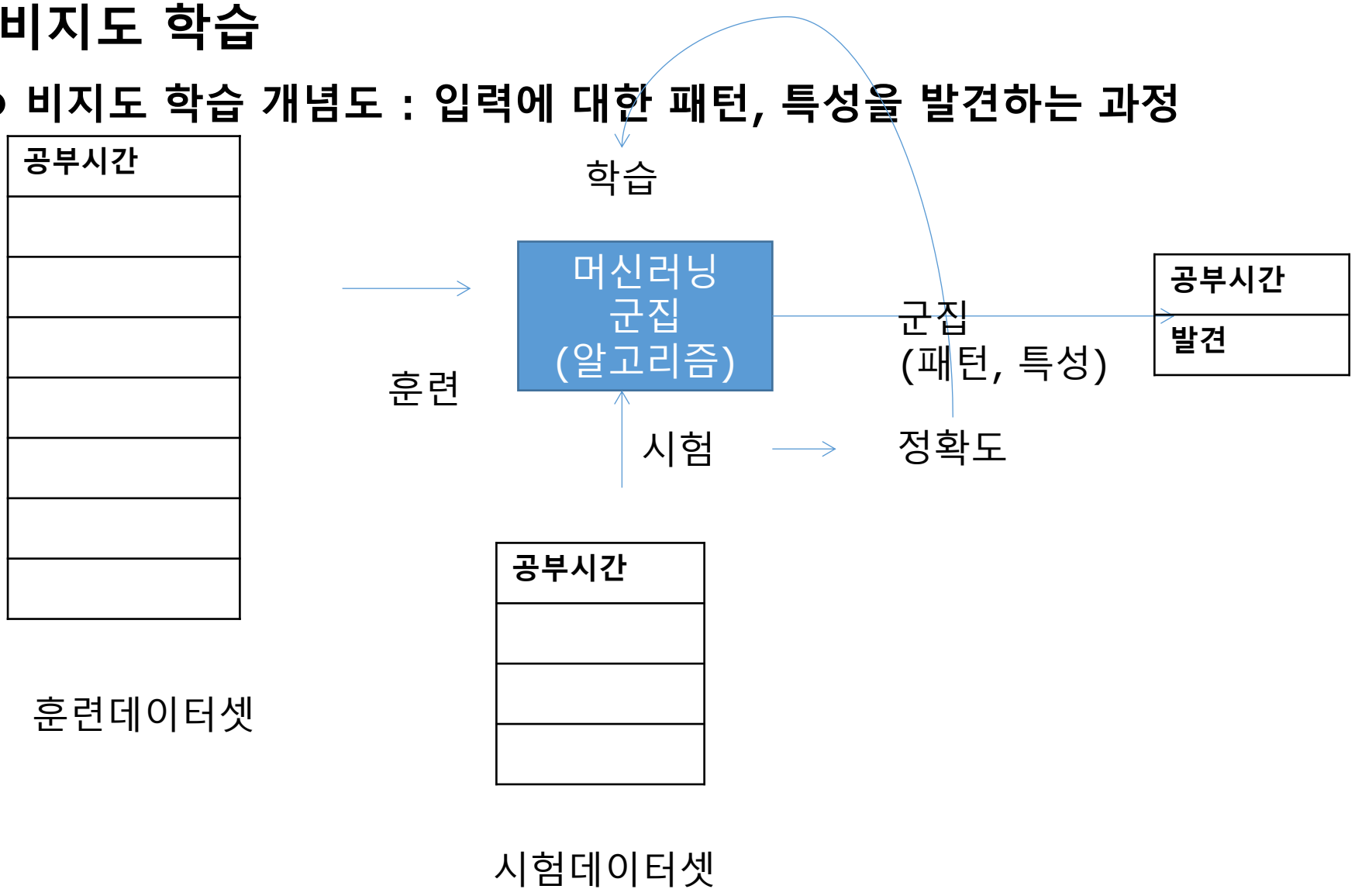


<자율학습의 예>

1. 머신러닝(Machine Learning)

비지도 학습

- 비지도 학습 개념도 : 입력에 대한 패턴, 특성을 발견하는 과정



1. 머신러닝(Machine Learning)

머신러닝 분류

머신러닝	지도학습	회귀	숫자값 예측	기존 온도 추이를 보고 내일 온도 예측	
		분류	항목 분류	문서 분류	
		추천 시스템과 랭킹학습	선호도 예측	영화 추천	
	비지도학습	군집화와 토픽 모델링	군집화: 비슷한 데이터들을 묶어서 큰 단위로 만드는 기법 토픽 모델링: 텍스트 데이터에 대한 토픽 관련 정도 표현	기사에 대한 스포츠 혹은 경제 클러스터로 구분	
		밀도 추정	관측 데이터로 부터 원래의 분포 추측	키와 몸무게 통계 자료에서 키와 몸무게의 관계 분석	커널 밀도 추정, 가우스 혼합 모델
		차원 축소	데이터의 차원을 낮추는 기법		주성분 분석(PCA), 특잇값 분해
	강화학습		기계가 환경과의 상호작용을 통해 장기적으로 얻는 이득을 최대화하도록 하는 학습 방법		

1. 머신러닝(Machine Learning)

머신러닝 분류

머신러닝	지도학습	회귀	숫자값 예측	기존 온도 추이를 보고 내일 온도 예측	
		분류	항목 분류	문서 분류	
		추천 시스템과 랭킹학습	선호도 예측	영화 추천	
	비지도학습	군집화와 토픽 모델링	군집화: 비슷한 데이터들을 묶어서 큰 단위로 만드는 기법 토픽 모델링: 텍스트 데이터에 대한 토픽 관련 정도 표현	기사에 대한 스포츠 혹은 경제 클러스터로 구분	
		밀도 추정	관측 데이터로 부터 원래의 분포 추측	키와 몸무게 통계 자료에서 키와 몸무게의 관계 분석	커널 밀도 추정, 가우스 혼합 모델
		차원 축소	데이터의 차원을 낮추는 기법		주성분 분석(PCA), 특잇값 분해
	강화학습		기계가 환경과의 상호작용을 통해 장기적으로 얻는 이득을 최대화하도록 하는 학습 방법		

1. 머신러닝(Machine Learning)

문제 유형에 따른 분류

문제유형	회귀문제	입력을 받아서 가장 적합한 숫자값 예측	주가 예측	선형 회귀, 확장된 회귀분석(ex : 다항회귀, 비선형 회귀, 벌점화 회귀 등),가우시안 프로세스 회귀, 칼만 필터, 인공 신경망 분석(Artificial Neural Network) ,의사결정트리(Decision Tree), 서포트 벡터 머신(회귀) (Support Vector Machine (Regression)), PLS(Partial Least Squares), 앙상블 기법(랜덤 포레스트 등)
	분류문제	주어진 입력에 대해 선택지 선택	단어로 기사 분류	로지스틱 회, 인공 신경망 분석(Artificial Neural Network) , CRF, RNN, 의사결정트리(Decision Tree) 서포트 벡터 머신(Support Vector Machine), 나이브 베이즈(Naive Bayes) , 앙상블 기법(랜덤 포레스트 등)
	군집화	주어진 입력을 비슷한 것끼리 군집	비슷한 문서 군집	K-means clustering, mean shift ,LDA(latent dirichlet allocation)
	차원축소 기법,			
	연관관계분석			
	자율학습 인공 신경망 (SOM 등)			
	표현형학습	풀고자 하는 문제에 적합한 표현 추출	일기를 통해 글쓴이의 감정 파악	딥러닝(단어 임베딩) 방법 : word2vec, 행렬 분해

1. 머신러닝(Machine Learning)

머신러닝의 주요 도전 과제

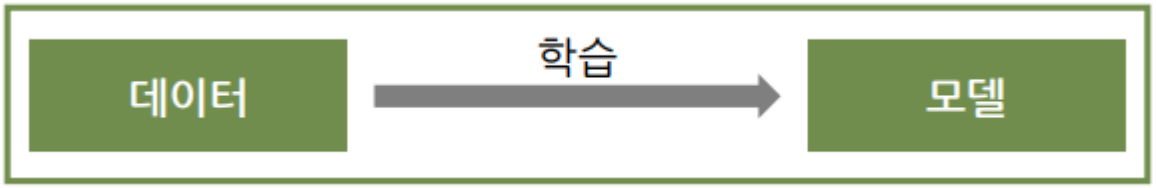
- 머신러닝의 주요 작업은 학습 알고리즘을 선택해서 어떤 데이터에 훈련시키는 것
  - '나쁜 데이터'
    - 충분하지 않은 양의 훈련 데이터
    - 대표성 없는 훈련 데이터
    - 낮은 품질의 데이터
    - 관련 없는 특성
  - '나쁜 알고리즘'
    - 훈련 데이터 과대적합
    - 훈련 데이터 과소적합

1. 머신러닝(Machine Learning)

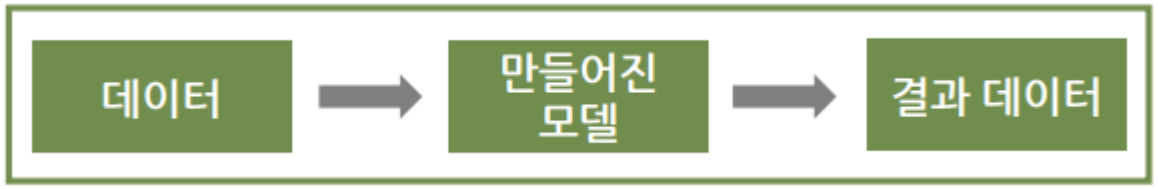
머신러닝 프로젝트 절차

- 품목코드 데이터와 … 데이터를 머신러닝 프로젝트 프로세스 검토

- 데이터로부터 모델을 만드는 단계

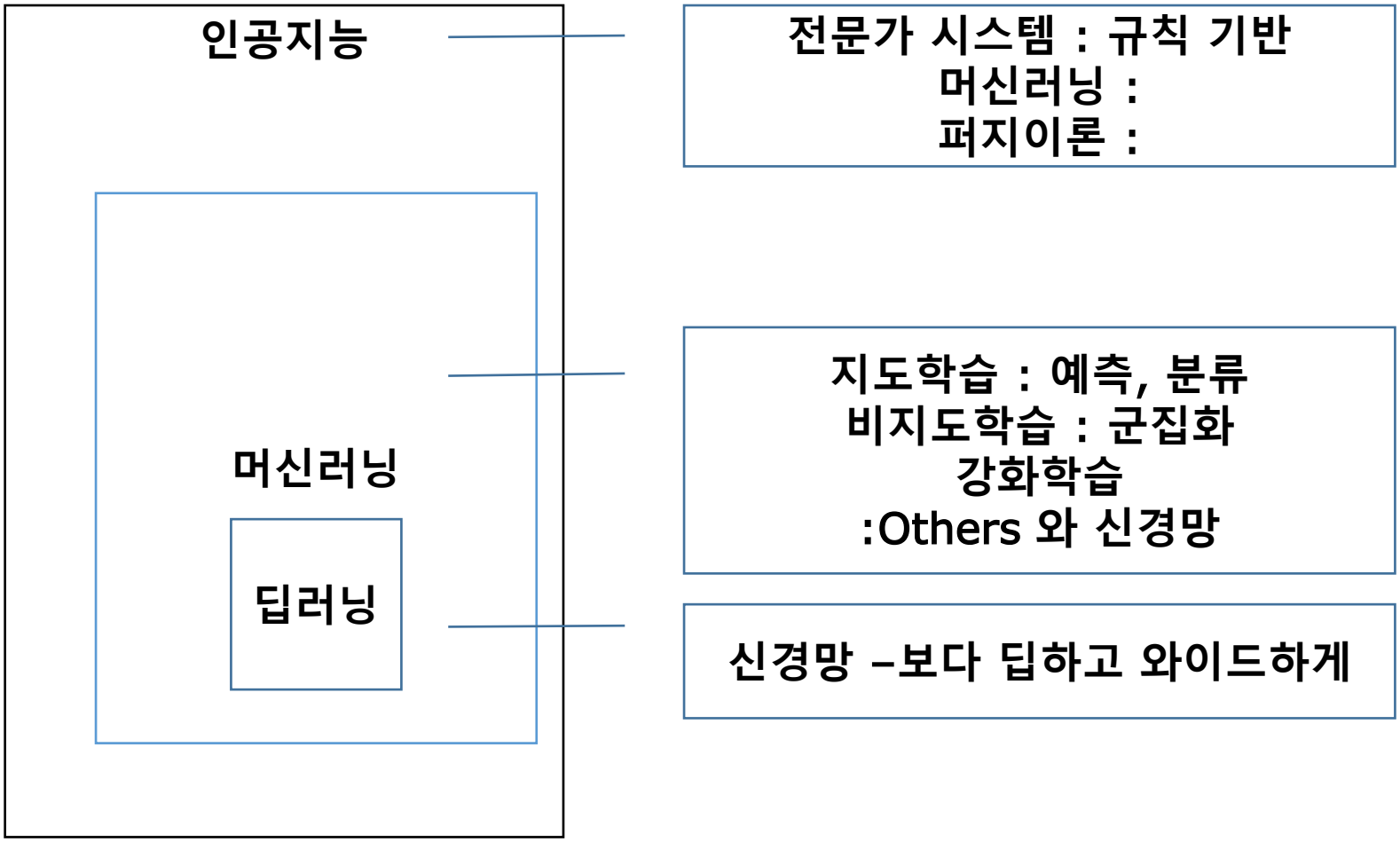


- 만들어진 모델을 적용하는 단계



1. 머신러닝(Machine Learning)

인공지능 머신러닝, 딥러닝 개념도



2. 지도학습

지도학습?

- 지도 학습(Supervised Learning)은 기계 학습의 한 유형으로, 입력 데이터와 해당 데이터에 대한 정확한 출력(레이블 또는 타겟) 사이의 관계를 모델링하는 방법입니다. 이 방법은 주어진 입력에 대한 출력을 예측하거나 분류하는 데 사용됩니다.
- 지도 학습은 다음과 같은 주요 특징을 갖고 있습니다.

1. 입력과 출력 데이터가 쌍을 이룸: 훈련 데이터에는 입력과 해당하는 정확한 출력이 함께 제공됩니다.
2. 모델 학습: 모델은 입력과 출력 사이의 패턴이나 관계를 학습하고 이를 나타내는 함수를 만듭니다. 주어진 입력으로부터 정확한 출력을 예측하기 위해 모델은 훈련 데이터에서 학습된 패턴을 기반으로 작동합니다.
3. 예측 또는 분류: 모델이 학습한 관계를 사용하여 새로운 입력에 대한 출력을 예측하거나 분류합니다.
4. 평가: 모델의 성능은 테스트 데이터나 실제 환경에서의 성능으로 평가됩니다.

- 지도 학습에는 회귀(Regression)와 분류(Classification)가 포함됩니다.

1. 회귀는 연속적인 값의 출력을 예측하는 것으로, 예를 들어 주택 가격이나 온도 예측과 같은 작업에 사용됩니다.
2. 분류는 주어진 입력을 여러 클래스 중 하나로 분류하는 것으로, 이메일 스팸 필터링이나 손글씨 숫자 인식과 같은 작업에 활용됩니다.

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- k-NN은 가장 단순한 예측 모델 중 하나이다. 유일하게 필요한 것은 다음과 같다.
  - 거리를 재는 방법
  - 서로 가까운 점들은 유사하다는 가정
- K-최근접 이웃(K-Nearest Neighbor) 기법은 목표변수의 범주를 알지 못하는 데이터 세트의 분류를 위해 해당 데이터 세트와 가장 유사한 주변 데이터 세트의 범주로 지정하는 방식으로 분류예측을 하는 기법이다. 이러한 K-최근접 이웃 기법에서는 해당 데이터 점과 주변 데이터 세트 간의 '유사성'을 어떻게 측정할 것인가와 최종적으로 목표변수의 범주를 분류할 때 주변 데이터 세트 몇 개를 기준으로 판단할 것인가에 대한 기준이 필요하다.
- 유사성 측정 방법
  - 데이터 간의 유사성을 측정하는 방식은 여러 가지가 있지만, 일반적으로 두 점간의 유클리디안 제곱 거리의 역수를 취하거나 피어슨 상관계수를 이용하여 유사성을 계산한다. 점들이 이산형 변수일 경우 자카드 계수(Jaccard coefficient)를 사용한다.

2. 분류 및 회귀

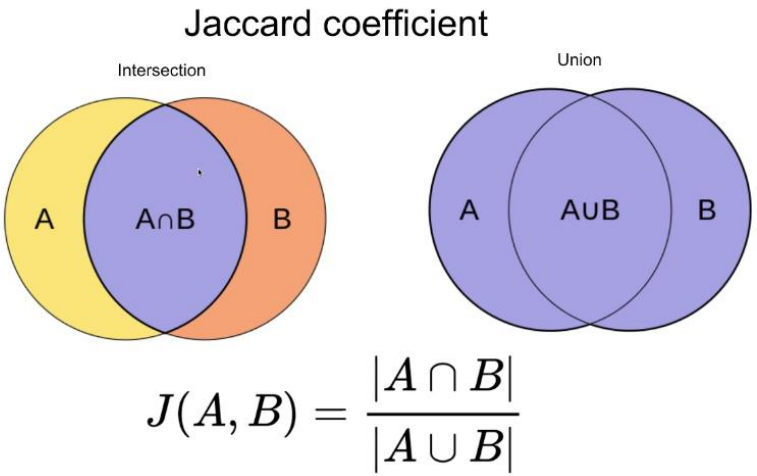
k-Nearest Neighbor (k-NN)

- 목표변수 분류 기준
  - K-최근접 이웃에서의 'K'는 해당 데이터 점과 주변 데이터 세트 간의 유사성을 측정한 후, 해당 데이터 점의 목표변수 분류를 위해 참조할 주변 데이터 점들의 개수를 의미한다. 예를 들어 어떤 고객이 좋아하는 영화와 유사하면서 기존에 시청하지 않았던 영화에 대한 추천을 생각해볼 수 있다. 이럴 때 해당 고객이 좋아하는 영화와 가장 유사한 영화 1개만 고려한다면 '1-근접이웃'이라고 말할 수 있고, 유사한 영화 3개를 고려한다면 '3-근접이웃'이라고 말할 수 있다. 이런 식으로 해당 데이터 점과 유사한 k개의 주변 데이터 점을 살펴보고, 해당 데이터 점의 분류범주를 고려하여 다수결의 원칙에 따라 새로운 범주를 결정하는 방식이 K-근접이웃 기법이다.

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- (0,0)을 분모, 분자에서 고려하지 않고 상관계수를 구함

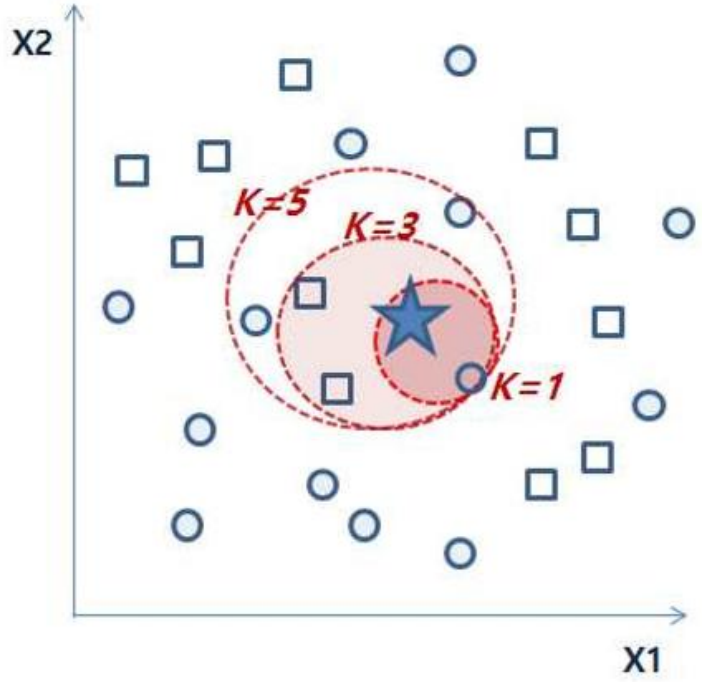


  - (0,0)을 제외해주는 것은 왜 중요할까요?
  - '행동이 발생하지 않았다면, 결과가 발생하지 않는다'의 조건이 반영하기 때문입니다. 우리는 행동에 따른 결과를 알고 싶은데 0을 제외하지 않게 된다면, 'A: 행동이 발생하지 않았고 결과도 발생하지 않았다. 그렇기 때문에 A는 관계가 있다'는 모순적인 내용이 분석 결과에 포함됩니다.

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- [그림]은 K-최근접 이웃 기법에서의 K 값 설정에 따른 목표 변수값 변화를 개념적으로 표현하고 있다. '☆'으로 표시된 점이 새롭게 분류해야 할 데이터 값이 고 나머지'□'와 '○'로 표시된 점들은 주변에 존재하는 다른 데이터 점이다.



K-최근접 이웃 기법에서의 K 값 변화에 따른 목표변수 범주 변화

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- 여기서 K=1로 설정하면, '☆'와 가장 가까운 데이터 점은'◯'이므로'☆'의 목표변 수는 '◯'로 분류될 것이다. 반면 K=3으로 설정하면'☆'와 가장 가까운 3개의 점 을 고려하게 되므로'□'가 2개,'◯'가 1개이므로 이때는 '☆'점은'□'로 분류 되게 된다.
- 한편 K=5로 설정한다면,'☆'주변에'◯'가 더 많으므로, 다수결의 원칙 에 의하여 또다시 '◯'로 분류될 것이다. 여기서 우리가 알 수 있는 것은 K-최근접 이웃 기법에서는 K 값을 어떻게 정하느냐에 따라 목표변수의 범주 예측 결과가 크게 달라질 수 있다는 점이다. 따라서 K-최근접 이웃 기법에서는 적절한 K 값을 정하는 것이 중요하다.
- 그러나 적절한 K 값에 대하여 이론적이나 통계적으로 명확한 기준이 있는 것은 아니며, 대개 여러 가지 K 값을 설정해보면서 반복적으로 테스트하여 최적의 분류성능을 보이는 K 값을 최종적으로 정하게 된다. 일반적으로는 K=3에서 K=9 사이의 범위 내에서 임의로 최초 K 값을 정한 뒤 훈련 데이터와 평가 데이터를 이용하여 분류성능을 테스트해보면서 최적 K 값을 정하는 방법을 사용하기도 한다.

k-Nearest Neighbor (k-NN)

- K-최근접 이웃 기법의 장/단점

장 점	단 점
✓ 알고리즘 이해가 쉽고 직관적이다. ✓ 사전 모형 설정 및 모수 추정이 필요 없다. ✓ 데이터 세트의 확률분포 등에 대한 가정이 필요 없다. ✓ 빠른 훈련(학습) 시간	✓ 모형이 없으므로, 변수들간의 관계 등 가설검증이나 구조 등에 대한 분석을 통해 통찰력을 얻기 어렵다. (특정한 가설이나 모형 없이 주어진 데이터를 통해 범주의 분류결과만 판단) ✓ K에 대한 명확한 기준 없으며 시행착오적 접근 필요 ✓ 느린 분류(예측) 소요시간 (새로운 데이터가 주어질 때마다 모든 데이터와의 유사성 계산해야 하므로 그만큼 시간 소요. 이런 특성으로 인해 게으른 학습(Lazy Learning)으로 불림) ✓ 많은 메모리 소요(대용량 데이터일 때 불리함)

2. 클러스터링(군집)

k-Nearest Neighbor (k-NN)

- **K-최근접 이웃 기법의 활용 분야**
  - K-최근접 이웃 기법의 몇 가지 단점에도 불구하고, 실제 비즈니스 실무현장에서는 매우 다양한 분야에 빈번하게 활용되고 있다. 특히 최근의 온라인 및 모바일 서비스 추천시스템에서 K-최근접 이웃 기법에 근거한 상품 및 서비스 추천 등이 대표적인 활용 분야라고 할 수 있다.

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- **분류 문제 예제:**

1. Iris 데이터셋을 로드하고, 특성과 타겟을 X와 y로 할당합니다.
2. StandardScaler()를 사용하여 특성을 표준화합니다.
3. KNeighborsClassifier()를 사용하여 KNN 분류 모델을 생성하고 훈련합니다.
4. 두 개의 특정 샘플(observation)에 대해 predict()를 사용하여 해당하는 클래스를 예측합니다.
5. predict_proba()를 사용하여 각 클래스에 속할 확률을 예측합니다.

k-Nearest Neighbor (k-NN)

- 분류 문제 예제:

```
# 라이브러리를 임포트
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

# 데이터 로드-아이리스
iris=datasets.load_iris()
X=iris.data
y=iris.target

# 표준화 객체 생성 - 특성 파악을 위한 작업
sd=StandardScaler()

# 특성을 표준화 함 - 특성 파악을 위한 작업
X_standardized=sd.fit_transform(X)

# k=5인 최근적 이웃 모델 생성-꽃받침, 꽃잎
knn=KNeighborsClassifier(n_neighbors=5, n_jobs=-1).fit(X_standardized, y)
```

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- **분류 문제 예제:**

```
# 시험을 위한 2개의 샘플 생성
observation=[
        [0.75, 0.75, 0.75, 0.75,],
        [1,1,1,1]
]


#  2개의 샘플이 속할 클래스 예측
knn.predict(observation)
>>
array([1, 2])
```

- knn.predict() 함수는 K-최근접 이웃 분류기를 사용하여 새로운 샘플에 대한 클래스를 예측합니다. 첫 번째 샘플은 [0.75, 0.75, 0.75, 0.75]로, 두 번째 샘플은 [1, 1, 1, 1]로 주어졌습니다.
- 그 결과, 예측된 클래스는 array([1, 2])로 나타났습니다. 이것은 첫 번째 샘플이 클래스 1에, 두 번째 샘플이 클래스 2에 속한다는 것을 의미합니다. 클래스는 0부터 시작하는 인덱스로 나타내어졌으며, 0, 1, 2 중 하나의 클래스에 해당하는 것입니다.

k-Nearest Neighbor (k-NN)

- **분류 문제 예제:**

```
# 2개의 샘플이 세 클래스에 속할 확률을 조회

knn.predict_proba(observation)

>>
array([[0. , 0.6, 0.4],
       [0. , 0. , 1. ]])
```

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- 회귀 문제 예제:

1. make_regression()을 사용하여 가상의 회귀 데이터를 생성합니다.
2. train_test_split()을 사용하여 데이터를 훈련 세트와 테스트 세트로 분할합니다.
3. KNeighborsRegressor 객체를 생성하고, n_neighbors 매개변수를 조정하여 이웃의 개수를 설정합니다.
4. fit() 메서드를 사용하여 모델을 훈련 세트에 맞춥니다.
5. predict()를 사용하여 테스트 세트에 대한 예측을 수행합니다.
6. 예측값과 실제값 사이의 평균 제곱 오차(MSE)를 계산하여 모델의 성능을 평가합니다.

k-Nearest Neighbor (k-NN)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 데이터 생성 (임의의 회귀 데이터 생성)
X, y = make_regression(n_samples=100, n_features=1, noise=0.1, random_state=42)

# 훈련 세트와 테스트 세트로 데이터를 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# K-최근접 이웃 회귀 모델 생성
knn_regressor = KNeighborsRegressor(n_neighbors=5)

# 모델을 훈련 세트에 맞춤
knn_regressor.fit(X_train, y_train)

# 모델을 사용하여 테스트 세트에 대한 예측 수행
y_pred = knn_regressor.predict(X_test)

# 예측값과 실제값 사이의 평균 제곱 오차(MSE) 계산
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
>>
Mean Squared Error: 2.5881038103118676
```

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

- 최선의 이웃 개수 결정하기

◦ 최적의 이웃 개수(k)를 찾기 위해 그리드 서치(Grid Search)를 수행하는 예제입니다. 그리드 서치는 여러 하이퍼파라미터 값들 중에서 최적의 값을 찾는 데 사용됩니다.

◦ 여기서 사용된 주요 단계는 다음과 같습니다.

1. 데이터 로드 및 전처리:Iris 데이터셋을 로드하고 특성(features)과 타겟(target)을 준비합니다.
2. 모델 및 파이프라인 설정:StandardScaler()를 사용하여 데이터를 표준화하는 전처리 단계(standardizer)를 설정합니다.
3. KNeighborsClassifier()를 사용하여 KNN 분류기(knn)를 설정합니다.
4. Pipeline을 사용하여 데이터를 표준화하고 KNN 분류기를 연결하여 전체 파이프라인(pipe)을 만듭니다.
5. 탐색할 하이퍼파라미터 후보 설정:탐색할 KNN 분류기의 이웃 개수(k)에 대한 후보값들을 지정합니다.

2. 분류 및 회귀

k-Nearest Neighbor (k-NN)

● 최선의 이웃 개수 결정하기

6. 그리드 서치(Grid Search) 수행:GridSearchCV를 사용하여 주어진 파라미터 그리드(search_space)를 기반으로 최적의 모델을 찾습니다.
7. cv=5는 5-폴드 교차 검증을 의미합니다.
8. fit()을 호출하여 최적의 하이퍼파라미터를 탐색하고, 교차 검증을 수행하여 모델을 훈련합니다.
9. 최적의 이웃 개수(k) 확인:best_estimator_.get_params()["knn__n_neighbors"]를 통해 최적의 이웃 개수(k)를 확인합니다.

- GridSearchCV는 지정된 파라미터 후보군을 탐색하면서 모델의 성능을 교차 검증을 통해 평가하고, 주어진 하이퍼파라미터 값 중에서 최적의 조합을 찾습니다. 이 경우, 최적의 이웃 개수(k)를 best_estimator_를 통해 확인할 수 있습니다.

k-Nearest Neighbor (k-NN)

- 최선의 이웃 개수 결정하기

```
# 라이브러리를 임포트합니다.
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.model_selection import GridSearchCV

# 데이터를 로드합니다.
iris = datasets.load_iris()
features = iris.data
target = iris.target

# 표준화 객체를 만듭니다.
standardizer = StandardScaler()

# KNN 분류기를 만듭니다.
knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
```

k-Nearest Neighbor (k-NN)

```
# 파이프라인을 만듭니다.
pipe = Pipeline([("standardizer", standardizer), ("knn", knn)])

# 탐색 영역의 후보를 만듭니다.
search_space = [{"knn__n_neighbors": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}]

# 그리드 서치를 만듭니다.
classifier = GridSearchCV(
    pipe, search_space, cv=5, verbose=0).fit(features, target)

# 최선의 이웃 개수 (k)
classifier.best_estimator_.get_params()["knn__n_neighbors"]

>>

6
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
  - 패키지 로드 및 데이터 셋 가져오기

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

df = pd.DataFrame(X, columns = iris.feature_names)
print("< Iris Data >")
print("The number of sample data : " + str(len(df)))
print("The number of features of the data : " + str(len(df.columns)))
print("The labels of the data : " + str(np.unique(y)))
df
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
  - 패키지 로드 및 데이터 셋 가져오기

>>

< Iris Data >
The number of sample data : 150
The number of features of the data : 4
The labels of the data : [0 1 2]

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
- 데이터 셋 분리

```
# split whole data set into train set and test set
# test_size : the proportion of the dataset to include in the test split. (0~1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,
                                                    random_state = 42)

print("The number of train data set : %d " %len(X_train))
print("The number of test data set : %d " %len(X_test))


>>
The number of train data set : 100
The number of test data set : 50
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
- 임의의 K 설정, 모델 학습 및 정확도 측정

```
# instantiate learning model (k = 3)
estimator = KNeighborsClassifier(n_neighbors=3)
# fitting the model
estimator.fit(X_train, y_train)
# predict the response
label_predict = estimator.predict(X_test)
# evaluate accuracy
print("The accuracy score of classification: %.9f"
      %accuracy_score(y_test, label_predict))


>>
The accuracy score of classification: 0.980000000
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
- 임의의 K 설정, 모델 학습 및 정확도 측정

```
# perform 10-fold cross validation

# create odd list of k for kNN
myList = list(range(1,50))
neighbors = [ x for x in myList if x % 2 != 0]
print(neighbors)
print("The number of neighbors k is %d" %len(neighbors))


>>
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49]
The number of neighbors k is 25
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
- 최적의 k값을 찾기 위한 Cross Validation 방법 사용

```
# empty list that will hold cross validation scores
cv_scores = []
# perform 10-fold cross validation
for k in neighbors:
    print("< k = %d >" %k)
    estimator = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(estimator, X_train, y_train, cv = 10, scoring = 'accuracy')
    print("The scores of classification are \n" + str(scores))
    cv_scores.append(scores.mean()) # average error
    print("The average score of scores is %.9f \n" %scores.mean())
```

```
>>
< k = 1 >
The scores of classification are
[1.  0.9 1.  0.8 0.8 1.  1.  1.  1.  0.9]
The average score of scores is 0.940000000

< k = 3 >
The scores of classification are
[0.9 1.  1.  0.7 0.9 1.  1.  1.  1.  0.9]
The average score of scores is 0.940000000

....
```

k-Nearest Neighbor (k-NN)

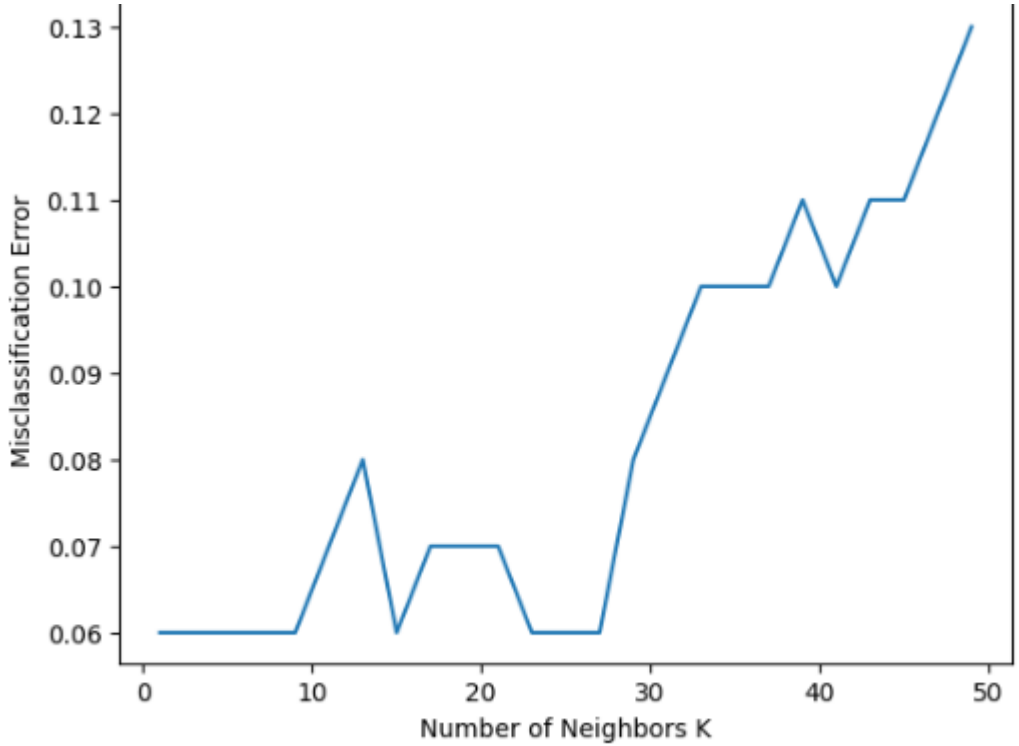
- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
- 최적의 k값을 찾기 위한 Cross Validation 방법 사용

```
# changing to misclassification rate (a.k.a classification error)
# MSE = 1 - cross validation score
MSE = [1 - x for x in cv_scores]

# plot misclassification error vs k
plt.plot(neighbors, MSE)
plt.xlabel("Number of Neighbors K")
plt.ylabel("Misclassification Error")
plt.show()

# determining best k
min_MSE = min(MSE)
index_of_min_MSE = MSE.index(min_MSE)
optimal_k = neighbors[index_of_min_MSE]
print ("The optimal number of neighbors i is %d" % optimal_k)
>>
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
  - 최적의 k값을 찾기 위한 Cross Validation 방법 사용

>>




```
The optimal number of neighbors i is 1
```

k-Nearest Neighbor (k-NN)

- Scikit-learn을 이용한 분류 예제 - Iris Data 분류
- 최적의 k값을 찾기 위한 Cross Validation 방법 사용

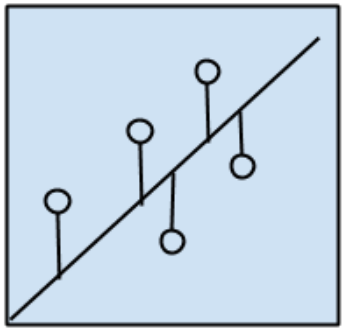
```
# instantiate learning model (k = 7)
estimator = KNeighborsClassifier(n_neighbors=3)
# fitting the model
estimator.fit(X_train, y_train)
# predict the response
label_predict = estimator.predict(X_test)
# evaluate accuracy
print("The accuracy score of classification: %.9f"
      %accuracy_score(y_test, label_predict))
```

```
>>

The accuracy score of classification: 0.980000000
```

2. 분류 및 회귀

선형 회귀분석- scikit-learn 패키지 사용

- 모델에 의한 예측을 위해 오차 측정을 사용하여 반복적으로 구체화된 변수 간의 관계를 모델링
- 통계의 핵심이며 통계 기반 기계학습에서 채택
- 알고리즘
  - Ordinary Least Squares Regression (OLSR)
  - Linear Regression
  - Logistic Regression
  - Stepwise Regression
  - Multivariate Adaptive Regression Splines (MARS)
  - Locally Estimated Scatterplot Smoothing (LOESS)



Regression Algorithms

출처: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

2. 분류 및 회귀

선형 회귀분석- scikit-learn 패키지 사용

- 패키지 로드 및 데이터 생성

```
from sklearn import linear_model
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.style.use('ggplot')

data = {'x1' : [13, 18, 17, 20, 22, 21],
        'x2' : [9, 7, 17, 11, 8, 10],
        'y' : [20, 22, 30, 27, 35, 32]}
data = pd.DataFrame(data)
X = data[['x1', 'x2']]
y = data['y']
data
>>
```

	x1	x2	y
0	13	9	20
1	18	7	22
2	17	17	30
3	20	11	27
4	22	8	35
5	21	10	32

2. 분류 및 회귀

선형 회귀분석- scikit-learn 패키지 사용

- 데이터 학습

```
linear_regression = linear_model.LinearRegression()
linear_regression.fit(X = pd.DataFrame(X), y = y)
prediction = linear_regression.predict(X = pd.DataFrame(X))
print('a value = ', linear_regression.intercept_)
print('b balue = ', linear_regression.coef_)
>>
a value =  -7.359201773835938
b balue =  [1.5443459  0.62472284]
```

선형 회귀분석- scikit-learn 패키지 사용

- 적합도 검증

```
linear_regression = linear_model.LinearRegression()
linear_regression.fit(X = pd.DataFrame(X), y = y)
prediction = linear_regression.predict(X = pd.DataFrame(X))
print('a value = ', linear_regression.intercept_)
print('b balue = ', linear_regression.coef_)
>>
count    6.000000e+00
mean     2.368476e-15
std      2.622371e+00
min     -3.399667e+00
25%      -1.987805e+00
50%       5.828714e-01
75%       1.415327e+00
max      3.385809e+00
Name: y, dtype: float64
```

선형 회귀분석- scikit-learn 패키지 사용

- 적합도 검증

```
SSE = (residuals**2).sum()
SST = ((y-y.mean())**2).sum()
R_squared = 1 - (SSE/SST)
print('R_squared = ', R_squared)
>>
R_squared =  0.796944017668523
```

결정계수 79.6%로 결과를 독립변수들이 종속변수에 상당한 영향을 주고 있음을 확인했습니다.

선형 회귀분석- scikit-learn 패키지 사용

- 성능 평가

```
from sklearn.metrics import mean_squared_error
print('score = ', linear_regression.score(X = pd.DataFrame(X), y=y))
print('Mean_Squared_Error = ', mean_squared_error(prediction, y))
print('RMSE = ', mean_squared_error(prediction, y)**0.5)

>>
score =  0.796944017668523
Mean_Squared_Error =  5.730691056910575
RMSE =  2.3938861829482567
```

2. 분류 및 회귀

Noise data

- 잡음Noise data
- 측정된 변수(속성)에서의 오류나 오차값
- 오류나 오차 값에 의해 경향 성훼손발생
- 잡음에 대한 훼손을 줄이기 위해 데이터 평활화기법 smoothing technique 존재
- 데이터 평활화 기법
  - 구간화Binning
  - 회귀Regression
  - 군집화Clustering

2. 분류 및 회귀

구간화Binning

- 정렬된 데이터 값들을 몇 개의 빈(혹은버킷)으로 분할하여 평활화하는 방법
- 이웃neighborhood(주변 값)들을 참조하여 정렬된 데이터를 매끄럽게함
  - 평균값 평활화smoothing by bin means : 빈bin에 있는 값들이 그 빈의 평균값mean으로 대체
  - 중앙값 평활화smoothing by bin medians : 빈bin에 있는 값들이 그 빈의 중앙값median으로 대체
  - 경계값 평활화smoothing by bin boundaries : 빈bin의 최소값과 최대값이 그 빈의 경계값이 되며, 두경 계값 중 가까운 쪽 값으로 대체

2. 분류 및 회귀

Noise data와 회귀Regression

- 회귀함수를 이용한 데이터 평활화 기법
  - 선형 회귀 분석linear regression analysis은 하나의 속성이다른하나의 속성을 예측하는데 이용할 수 있도록 선형관계를 찾음
  - 다중회귀분석multiple regression analysis은 두개 이상의 속성을 가지고 다른 속성을 예측하는데 이용할 수 있도록 단순 선형 회귀분석의 확장
  - 선형 회귀 분석 또는 다중 회귀 분석을 이용하여 평활화

2. 분류 및 회귀

Noise data와 회귀Regression

- 회귀분석(Regression Analysis) 분석 모델은 사실 머신러닝 기법이라기보다는 통계학 영역에서 오랫동안 사용되어온 전형적인 통계분석 기법이라고 할 수 있다. 사실 대부분의 고급 통계 분석 기법들이 회귀분석적인 접근방법의 확장 또는 응용이라고 해도 과언이 아닐 정도로 회귀분석이 통계학의 영역에서 차지하는 비중은 막대하다고 할 수 있다.
- 머신러닝 관점에서는 수치예측을 위해 통계학의 회귀분석기법을 사용하여 훈련데이터를 학습시킨 뒤, 평가데이터를 통해 결과값을 예측한다는 점에서 지도학습으로 회귀분석 기법을 다루고 있다.
- 회귀분석의 모델 식

$$y = f(x) + \epsilon$$

  - ϵ 는 오차항으로서 기댓값이 0이고 분산이 σ2 이라고 가정한다. 회 귀분석에서는 독립변수 X가 주어졌을 때 목적변수(혹은 반응변수) Y의 조건부 기댓값을 특정 형태의 함수 f(·)로 가정하고 데이터 세트로부터 해당 함수 f(·)를 추정하게 된다.
- 선형회귀모형(linear regression model)

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D = w_0 + w^T x$$

2. 분류 및 회귀

Noise data와 회귀Regression

- sklearn 패키지의 datasets 서브패키지는 회귀분석을 공부하기 위한 예제를 제공한다. 보스턴의 506개 타운(town)의 13개 독립변수값로부터 해당 타운의 주택가격 중앙값을 예측하는 문제다.

◦ 사용할 수 있는 특징 데이터는 다음과 같다.

– 독립변수

① CRIM: 범죄율
② INDUS: 비소매상업지역 면적 비율
③ NOX: 일산화질소 농도
④ RM: 주택당 방 수
⑤ LSTAT: 인구 중 하위 계층 비율
⑥ B: 인구 중 흑인 비율
⑦ PTRATIO: 학생/교사 비율
⑧ ZN: 25,000 평방피트를 초과 거주지역 비율
⑨ CHAS: 찰스강의 경계에 위치한 경우는 1, 아니면 0
⑩ AGE: 1940년 이전에 건축된 주택의 비율
⑪ RAD: 방사형 고속도로까지의 거리
⑫ DIS: 직업센터의 거리
⑬ TAX: 재산세율

– 종속변수

① 보스턴 506개 타운의 1978년 주택 가격 중앙값 (단위 1,000 달러)

2. 분류 및 회귀

Noise data와 회귀Regression

- boston 데이타가 deprecated 되어 다음처럼 데이터 로드

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	import pandas as pd import numpy as np import seaborn as sns import matplotlib.pyplot as plt data_url = "http://lib.stat.cmu.edu/datasets/boston" raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None) data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]]) target = raw_df.values[1::2, 2] data
결과값1	array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02, 4.9800e+00], [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02, 9.1400e+00], [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02, 4.0300e+00], ..., [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02, 5.6400e+00], [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02, 6.4800e+00], [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02, 7.8800e+00]])
결과값2	
비고	

2. 분류 및 회귀

Noise data와 회귀Regression

- boston 데이타가 deprecated 되어 다음처럼 데이터 로드

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	target
결과값1	array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. , 18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6, 15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2, 13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7, 21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9, 35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5, 19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. , 20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2, 23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8, ... 20.6, 21.2, 19.1, 20.6, 15.2, 7. , 8.1, 13.6, 20.1, 21.8, 24.5, 23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
결과값2	
비고	

2. 분류 및 회귀

Noise data와 회귀Regression

- 분석할 데이터는 pandas 데이터프레임 형태로 만들어야 한다. 여기에서는 독립변수 행렬을 dfX로, 종속변수 벡터를 dfy로 만든다. 종속변수의 이름은 MEDV로 지정한다.
- 독립변수과 종속변수 데이터프레임을 하나의 데이터프레임으로 묶어두면 편리하다.

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	x =pd.DataFrame(data, columns=['CRIM','ZN','INDUS','CHAS', 'NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']) dfX =x[['CRIM','ZN','INDUS','CHAS', 'NOX','RM','AGE']] dfX
결과값1	CRIM ZN INDUS CHAS NOX RM AGE 0 0.00632 18.0 2.31 0.0 0.538 6.575 65.2 1 0.02731 0.0 7.07 0.0 0.469 6.421 78.9 2 0.02729 0.0 7.07 0.0 0.469 7.185 61.1 3 0.03237 0.0 2.18 0.0 0.458 6.998 45.8 4 0.06905 0.0 2.18 0.0 0.458 7.147 54.2 ... ... 501 0.06263 0.0 11.93 0.0 0.573 6.593 69.1 502 0.04527 0.0 11.93 0.0 0.573 6.120 76.7 503 0.06076 0.0 11.93 0.0 0.573 6.976 91.0 504 0.10959 0.0 11.93 0.0 0.573 6.794 89.3 505 0.04741 0.0 11.93 0.0 0.573 6.030 80.8
비고	

2. 분류 및 회귀

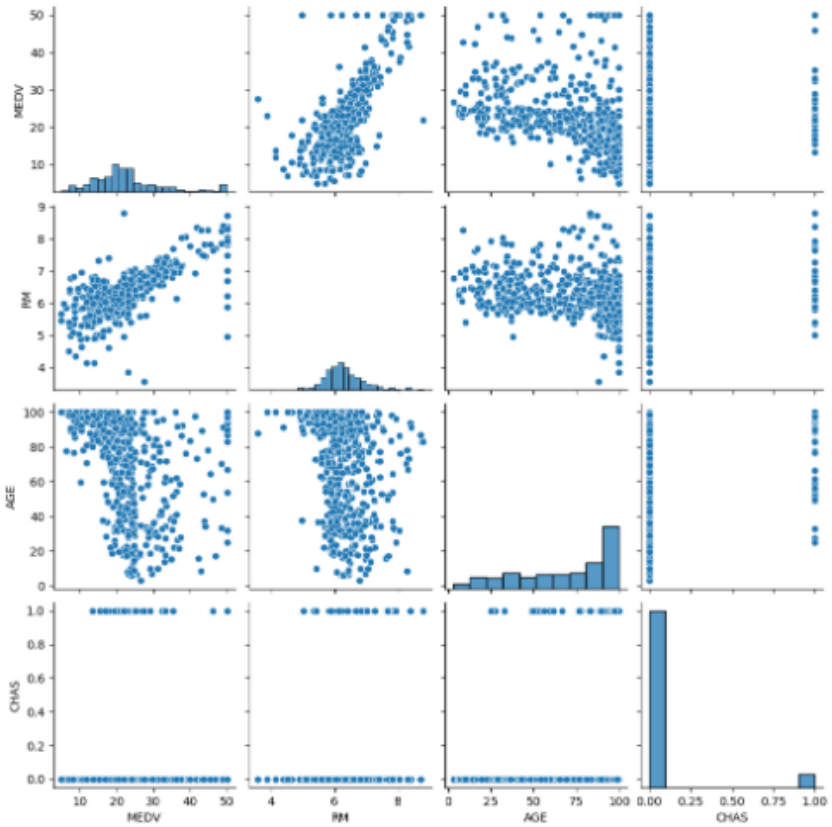
Noise data와 회귀Regression

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	dfy = pd.DataFrame(target, columns=['MEDV']) df = pd.concat([dfX, dfy], axis=1) df.tail()
결과값1	CRIM ZN INDUS CHAS NOX RM AGE MEDV 501 0.06263 0.0 11.93 0.0 0.573 6.593 69.1 22.4 502 0.04527 0.0 11.93 0.0 0.573 6.120 76.7 20.6 503 0.06076 0.0 11.93 0.0 0.573 6.976 91.0 23.9 504 0.10959 0.0 11.93 0.0 0.573 6.794 89.3 22.0 505 0.04741 0.0 11.93 0.0 0.573 6.030 80.8 11.9
비고	

2. 분류 및 회귀

회귀 모델 사례 -보스턴 집값

- 일부 독립변수와 종속변수의 관계를 스캐터플롯(scatter plot)으로 살펴본다.

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	sns.pairplot(dfX) plt.show()
결과값1	
비고	

2. 분류 및 회귀

회귀 모델 사례 -보스턴 집값

- 플롯의 첫 행을 보면 종속변수인 집값(MEDV)과 방 개수(RM), 노후화 정도(AGE)와 어떤 관계를 가지는지 알 수 있다.

  - 방 개수가 증가할 수록 집값은 증가하는 경향이 뚜렷하다.
  - 노후화 정도와 집값은 관계가 없어 보인다.

- 스케터플롯의 모양으로부터 찰스강 유역 여부(CHAS)는 범주값이며 값이 1이면 0일 때 보다 집값의 평균이 더 높아지는 것도 볼 수 있다.

2. 분류 및 회귀

Question 1

- 당뇨병 진행도 예측
  - scikit-learn 패키지가 제공하는 당뇨병 진행도 예측용 데이터는 442명의 당뇨병 환자를 대상으로한 검사 결과를 나타내는 데이터이다.
  - 이 데이터의 독립변수를 조사하고 어떤 데이터들이 주택가격과 상관관계가 있는지를 조사한다. 또한 서로 강한 상관관계를 가지는 독립변수도 알아보자.
  - 10 종류의 독립변수를 가지고 있다. 독립변수의 값들은 모두 스케일링(scaling)되었다.
    - age: 나이
    - sex: 성별
    - bmi: BMI(Body mass index)지수
    - bp: 평균혈압
    - s1~s6: 6종류의 혈액검사수치
  - 종속변수는 1년 뒤 측정한 당뇨병의 진행률이다.

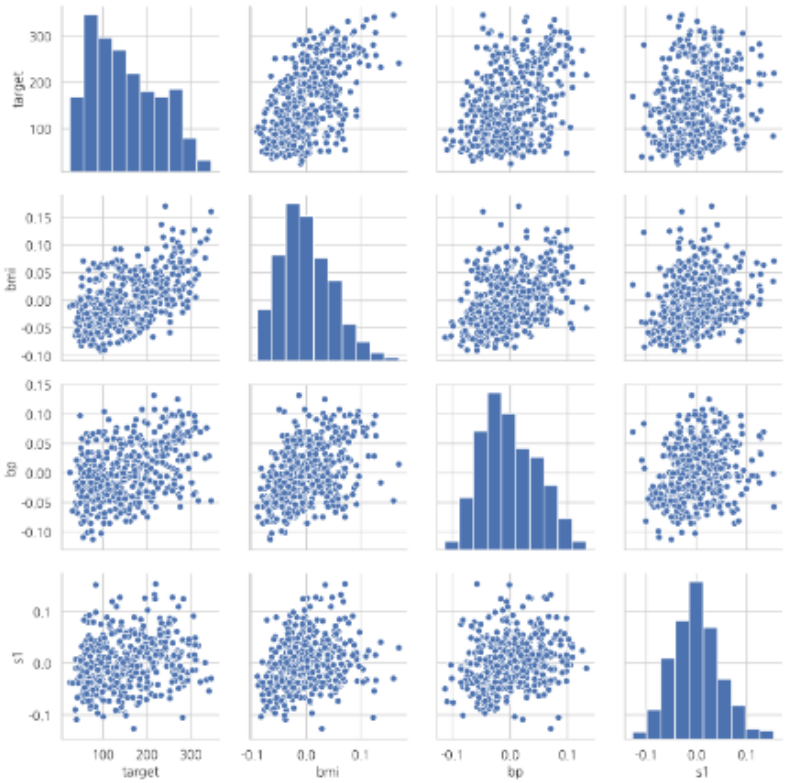
2. 분류 및 회귀

Answer 1

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	from sklearn.datasets import load_diabetes diabetes = load_diabetes() df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names) df["target"] = diabetes.target df.tail()
결과값1	age sex bmi bp s1 s2 s3 s4 s5 s6 target 437 0.041708 0.050680 0.019662 0.059744 -0.005697 -0.002566 -0.028674 -0.002592 0.031193 0.007207 178.0 438 -0.005515 0.050680 -0.015906 -0.067642 0.049341 0.079165 -0.028674 0.034309 -0.018114 0.044485 104.0 439 0.041708 0.050680 -0.015906 0.017293 -0.037344 -0.013840 -0.024993 -0.011080 -0.046883 0.015491 132.0 440 -0.045472 -0.044642 0.039062 0.001215 0.016318 0.015283 -0.028674 0.026560 0.044529 -0.025930 220.0 441 -0.045472 -0.044642 -0.073030 -0.081413 0.083740 0.027809 0.173816 -0.039493 -0.004222 0.003064 57.0
비고	

2. 분류 및 회귀

Answer 1

파일	소스코드
실습환경	준비 py3_10_tf
소스코드	sns.pairplot(df[["target", "bmi", "bp", "s1"]]) plt.show()
결과값1	
비고	

2. 분류 및 회귀

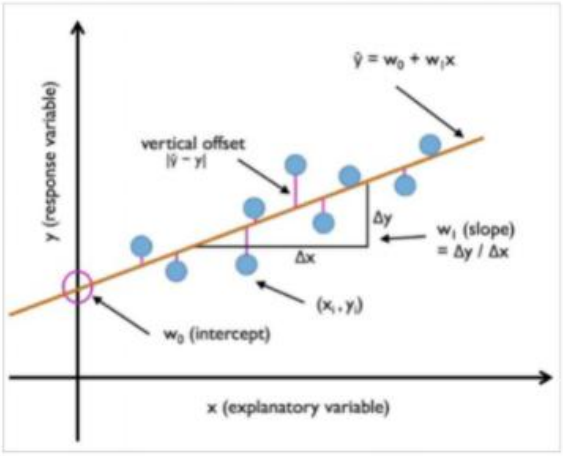
Answer 1

- 스케터플롯을 그려보면 독립변수인 BMI지수와 평균혈압이 종속변수인 당뇨병 진행도와 양의 상관관계를 가지는 것을 볼 수 있다. 또한 두 독립변수 BMI지수와 평균혈압도 서로 양의 상관관계를 가진다. 이렇게 독립변수끼리 상관관계를 가지는 것을 다중공선성(multicolinearity)이라고 한다. 다중공선성은 회귀분석의 결과에 영향을 미칠 수 있다.

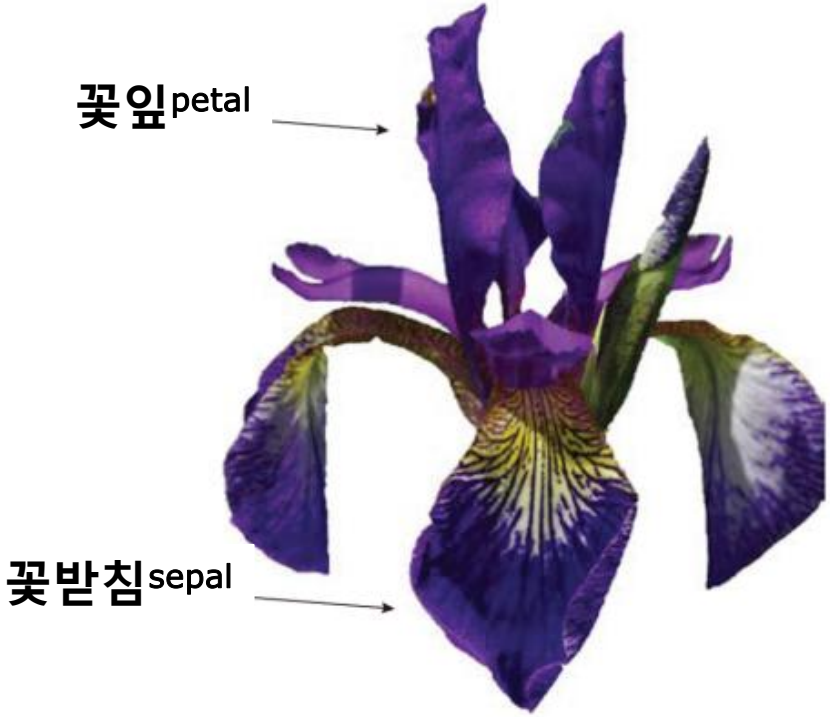
2. 분류 및 회귀

Question 2

- 붓꽃 예측
  - 붓꽃의 꽃잎petal과 꽃받침sepal의 폭과 길이를 센티미터 단위로 측정해 놓은 데이터를 통해, 새로 채집한 붓꽃의 품종을 예측하고자 한다.
  - '붓꽃(Iris)'은 프랑스의 국화



선형 회귀 그래프



2. 분류 및 회귀

Question 2

- 붓꽃 예측

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

data = datasets.load_iris()
#데이터셋
input_data = data['data'] # 꽃의  특징 (input data)
target_data = data['target'] #꽃  종류를  수치로 나탄내  것 (0 ~ 2) (target data)
flowers = data['target_names'] # 꽃  종류를  이름으로  나타낸  것
feature_names = data['feature_names'] # 꽃  특징들의  명칭
#sepal : 꽃받침
#petal : 꽃잎
print('꽃을  결정짓는  특징 :  {}'.format(feature_names))
print('꽃  종류 :  {}'.format(flowers))
```

2. 분류 및 회귀

Question 2

- **붓꽃 예측**

>>

```
꽃을 결정짓는 특징 :  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
꽃 종류 :  ['setosa' 'versicolor' 'virginica']
```

2. 분류 및 회귀

Question 2

- 붓꽃 예측

```
iris_df = pd.DataFrame(input_data, columns=feature_names)
iris_df['species'] = target_data
#맨 위에 있는 데이터 10개 출력
print(iris_df.head(10))
#데이터의 정보 출력
print(iris_df.describe())
```

>>

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	
5	5.4	3.9	1.7	0.4	
6	4.6	3.4	1.4	0.3	
7	5.0	3.4	1.5	0.2	
8	4.4	2.9	1.4	0.2	
9	4.9	3.1	1.5	0.1	

	species
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

2. 분류 및 회귀

Question 2

- 붓꽃 예측

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)     species
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

2. 분류 및 회귀

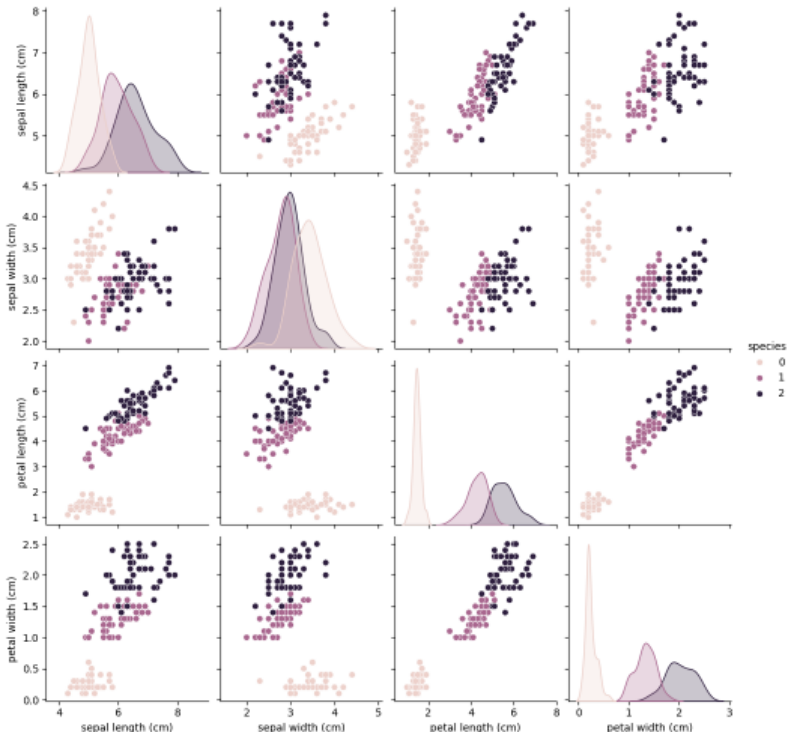
Question 2

- 붓꽃 예측

#4가지 변수(특징)의 관계를 'seaborn' 라이브러리에서 제공하는 pairplot() 메소드로 표현한 그래프 16가지

```
sns.pairplot(iris_df, hue='species', vars=feature_names)
plt.show()
```

>>



2. 분류 및 회귀

Question 2

- 붓꽃 예측

#각 특징들의 가중치(weight)와 절편(bias)을 확인

#로지스틱 회귀 모델의 가중치와 절편
#다중 분류 가중치와 절편을 출력하면, 각 클래스마다의 가중치 절편을 출력한다.
print(lr.coef_, lr.intercept_)

>>
[[-0.97511573 1.08893052 -1.78416098 -1.65224049]
 [0.5072161 -0.30353329 -0.3290721 -0.69052199]
 [0.46789963 -0.78539723 2.11323308 2.34276248]] [-0.39150253 1.92427457 -1.53277204]

- 첫번째 배열이 setosa에 대한 가중치와 절편이므로 해당 값들로 식을 구성해 보면,
- setosa(z1) = (-0.96 * sepal_length) + (1.09 * sepal_width) + (-1.78 * petal_length) + (-1.66 * petal_width) - 0.39

2. 분류 및 회귀

Question 2

- **붓꽃 예측**

◦ 경우의 수(클래스)가 3가지 이상이다.
◦ 이런 경우에는 시그모이드(Sigmoid) 함수가 아닌 소프트맥스(Softmax) 함수를 사용

$$e\_sum = e^{z1} + e^{z2} + e^{z3}$$

◦ z1은 setosa, z2는 versicolor, z3은 virginica의 방정식이다.
◦ 각 클래스들의 Z값을 모두 구한 후, 자연상수 e의 제곱으로 나타내어 모두 더한다. (= e_sum)

◦ 확률을 구하고 싶은 클래스의 e^z 을 전체 합으로 나누어주면,
◦ 해당 클래스의 확률이 되는 것이다. (= e^z / e_sum)
◦ setosa의 확률을 구하는 식

$$\text{setosa 확률} = \frac{e^{z1}}{e\_sum}$$

2. 분류 및 회귀

Question 2

- 붓꽃 예측

```
setosa_z1 = (-0.96 * 5.1) + (1.09 * 3.5) + (-1.78 * 1.4) + (-1.66 * 0.2) - 0.39
versicolor_z2 = (0.51 * 5.1) + (-0.30 * 3.5) + (-0.32 * 1.4) + (-0.7 * 0.2) - 1.92
virginica_z3 = (0.47 * 5.1) + (-0.79 * 3.5) + (2.11 * 1.4) + (2.34 * 0.2) - 1.53
print(setosa_z1)
print(versicolor_z2)
print(virginica_z3)

setosa_rs=setosa_z1/(setosa_z1+versicolor_z2+virginica_z3)
versicolor_rs=versicolor_z2/(setosa_z1+versicolor_z2+virginica_z3)
virginica_rs=virginica_z3/(setosa_z1+versicolor_z2+virginica_z3)
print(setosa_rs)
print(versicolor_rs)
print(virginica_rs)
```

2. 분류 및 회귀

Question 2

- 붓꽃 예측

>>

```
iris_df.head(3)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0

테스트

```
setosa_z1 = (-0.96 * 5.1) + (1.09 * 3.5) + (-1.78 * 1.4) + (-1.66 * 0.2) - 0.39
versicolor_z2 = (0.51 * 5.1) + (-0.30 * 3.5) + (-0.32 * 1.4) + (-0.7 * 0.2) - 1.92
virginica_z3 = (0.47 * 5.1) + (-0.79 * 3.5) + (2.11 * 1.4) + (2.34 * 0.2) - 1.53
print(setosa_z1)
print(versicolor_z2)
print(virginica_z3)
```

```
-4.294999999999999
-0.957
1.5239999999999994
```

```
setosa_rs=setosa_z1/(setosa_z1+versicolor_z2+virginica_z3)
versicolor_rs=versicolor_z2/(setosa_z1+versicolor_z2+virginica_z3)
virginica_rs=virginica_z3/(setosa_z1+versicolor_z2+virginica_z3)
print(setosa_rs)
print(versicolor_rs)
print(virginica_rs)
```

```
1.1520922746781115
0.25670600858369097
-0.40879828326180245
```

setosa가 확률 값이 가장 큼. 따라서 잘 분류 되고 있는 것을 알 수 있음

Question 2

- 붓꽃 예측

```
#decision_function()에 테스트 데이터 5개를 넣고 소수점 2자리까지 출력
#결정  함수(decision_function)로 z1 ~ z3의  값을  구한다.
decision = lr.decision_function(test_scaled[:5])
print(np.round(decision, decimals=2))
>>
[[-2.21  2.1   0.1 ]
 [ 5.87  2.56 -8.43]
 [-9.33  1.8   7.53]
 [-2.29  1.73  0.56]
 [-3.59  2.33  1.26]]
```

2. 분류 및 회귀

Question 2

- 붓꽃 예측

◦ softmax 함수
◦ z값으로 직접 확률을 구할 필요가 없다. decision_function()을 통해 구한 값을 scipy에서 제공하는 softmax 함수에 전달해주면 각 클래스에 대한 확률을 구해주기 때문이다.

```
#소프트맥스 함수를 사용한 각 클래스들의 확률
from scipy.special import softmax
proba = softmax(decision, axis=1)
print(np.round(proba, decimals=3))
>>
[[0.012 0.87  0.118]
 [0.965 0.035 0.   ]
 [0.    0.003 0.997]
 [0.013 0.752 0.234]
 [0.002 0.745 0.253]]
```

비교해 보자!!!

```
lr = LogisticRegression(max_iter=1000)

#로지스틱 회귀 학습
lr.fit(train_scaled, train_target)

#테스트 데이터 예측
pred = lr.predict(test_scaled[:5])
print(pred)
```

```
[1 0 2 1 1]
```

◦ 모델이 예측한 결과값 [1 0 2 1 1] 과 비교해보면,첫번째는 데이터는 87%의 확률로 1(versicolor)이라고 예상했고,두번째 데이터는 97%확률로 0(setosa)이라고 예상했다.

2. 분류 및 회귀

나이브 베이즈(Naive Bayes) 기법

- 베이즈 정리(Bayes'theorem)은 목표변수의 범주를 학습시키기 위해 통계학에서 사용되는 베이즈 정리를 사용한다. 즉, 나이브 베이즈 기법은 베이즈 확률 추정에 기반을 둔 확률모형이다. 이러한 나이브 베이즈 기법을 적용하기 위해서는 베이즈 정리와 사후확률의 개념을 이해할 필요가 있다.

◦ 베이즈 정리(Bayes'theorem)

- 베이즈 정리는 특정 사건이 발생할 확률을 다음과 같은 형태로 표현한 이론을 말한다.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} \qquad (3.1)$$

- 즉, 사건 B가 일어났을 때 사건 A가 일어날 확률(사후확률)은 사건 A가 일어날 확률(사전확률)과 사건 A가 일어났을 때 사건 B가 일어날 조건부 확률 P(B|A)의 곱을 사건 B가 일어날 확률 P(B)로 나누어 알아낼 수 있다는 뜻이다. 특히 분모에 있는 사건 B의 확률 P(B)는 다음과 같이 주변 확률(Marginal Probability)로 나타낼 수 있다.

$$P(B) = \sum_{A} P(A, B) = \sum_{A} P(B|A)P(A) \qquad (3.2)$$

나이브 베이즈(Naive Bayes) 기법

- 식 (3.1)의 분모에 식 (3.2)를 대입하면 (3.1)의 베이즈 정리는 다음과 같이 표현할 수 있다.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\sum_{A} P(B|A)P(A)} \qquad (3.3)$$

- 이처럼 베이즈 정리는 사전확률이 주어지고, 이에 따른 조건부확률 및 주변확률을 통해 조건과 결과를 서로 바꿔서 사후확률을 계산하는 공식이라고 할 수 있다.

2. 분류 및 회귀

나이브 베이즈(Naive Bayes) 기법

- 예를 들어 받은 이메일이 스팸일 확률을 구하는 문제를 고려해 보자.
  - 실제 데이터를 관측하기 전에 경험이나 주관적 확신 등에 의거하여, '내가 오늘 받은 이메일 중 스팸 메일이 포함되어 있을 가능성은 대략 10%일 것 같다.'고 가정할 수 있다. 이렇게 추가적인 증거 없이 가능성을 추정하는 것을 '사전확률'이라고 할 수 있다. 이제 실제 데이터를 관측했다고 가정해보자.
  - 오늘 도착한 이메일 중 스팸 메일들에 '반짝할인'이라는 단어가 들어가 있는 조건부 확률 P(반짝할인|스팸) 및 주변확률 P(반짝할인)의 확률을 계산할 수 있다. 이 경우 조건부확률 P(반짝할인 |스팸)를 0.4, 주변확률 P(반짝할인)를 0.05라고 가정하면, 본 예시에서 구하고자 하는 사후확률 P(스팸|반짝할인)는 (0.4*0.1)/0.05 = 0.8로 계산할 수 있다.
  - 즉, 받은 이메일에서 '반짝할인'이라는 단어가 들어가 있으면, 해당 이메일이 '스팸'일 확률은 80%라고 할 수 있다. 이는 데이터 세트를 관측하기 전의 사전확률 10%보다 8배 높아진 것이므로, '반짝할인'이라는 단어가 들어가 있는 이메일은 스팸 메일함으로 분류되어야 한다는 '강한 확신'을 줄 수 있다고 볼 수 있다.

2. 분류 및 회귀

나이브 베이즈(Naive Bayes) 기법

- 비록 단순한 예시지만, 매우 많은 단어 출현빈도들을 동시에 고려하면 바로 현실 문제에서 활용하는 스팸메일 필터의 원리와 거의 비슷한 분류기를 구현할 수 있다. 여기서 매우 많은 단어 빈도를 동시에 고려해야 하기 때문에, 모든 단어들이 출현할 가능성이 독립적이라고 가정하게 된다. 물론 이는 현실적이지 않다. 그렇기 때문에 이 기법이 'Naive'라고 불리는 것이다. 그러나 실제로는 이렇게 독립성 가정이 유효하지 않다고 판단될 때에도 실제로 나이브 베이즈 추정 결과는 여전히 결과를 잘 예측하는 것으로 알려져 있다. 그것이 가능한 이유는 나이브 베이즈에서는 해당 목표변수의 범주를 분류해내는 것이 목표인 것이지, 해당 분류에 속할 확률값의 정교함이 목적이 아니기 때문이다. 즉, 스팸메일 판정 규칙이 "사후확률값이 50% 이상일 때 스팸으로 판정한다." 일 경우 해당 확률 예측이 60%로 나왔든, 90%로 나왔든 모두 50% 이상이므로 결과적으로 두 경우 모두 스팸으로 분류해버리기 때문에 예측 확률값 자체가 정확하지 않아도 결과예측에는 큰 영향을 주지 않는다.

나이브 베이즈(Naive Bayes) 기법

● 나이브 베이즈 기법의 장/단점

장 점	단 점
✓ 개념과 이론이 단순하고 계산이 빠르다. ✓ 노이즈 및 결측치가 있어도 잘 수행된다. ✓ 고차원의 데이터 세트에 적합하다. ✓ 알고리즘의 단순성에 비해서 복잡한 분류 알고리즘보다 예측결과가 더 효과적인 경우 가 많다.	✓ 모든 속성이 동등하게 중요하고 독립적이라는 결함 가정에 의존한다. ✓ 독립변수들이 범주 형태가 아닌 수치 형태일 경우 정확성이 떨어진다. ✓ 범주 분류문제에는 적합하나, 예측된 범주의 확률값 을 활용해야 할 경우에는 적합하지 않다.

2. 분류 및 회귀

나이브 베이즈(Naive Bayes) 기법

- **나이브 베이즈 기법의 활용 분야**
  - 나이브 베이즈 기법은 위의 예시에서 제시한 것과 같이 실제로 스팸메일 필터에 많이 사용되고 있으며, 텍스트 데이터 등에 근거한 문서 분류에 많이 사용된다. 그 외에도 빠르고 효율적인 분류가 필요한 침입자 검출 및 이상행동 검출, 컴퓨터 네트워크 진단, 질병 진찰, 이탈 예측, 민원 예측 등의 분야에도 활용되고 있다.

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 패키지와 데이터 셋 로드

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

import pandas as pd
import numpy as np

tennis_data = pd.read_csv('datasets/playtennis.csv')
tennis_data
```

>>

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 데이터 전처리

```
tennis_data.Outlook = tennis_data.Outlook.replace('Sunny', 0)
tennis_data.Outlook = tennis_data.Outlook.replace('Overcast', 1)
tennis_data.Outlook = tennis_data.Outlook.replace('Rain', 2)

tennis_data.Temperature = tennis_data.Temperature.replace('Hot', 3)
tennis_data.Temperature = tennis_data.Temperature.replace('Mild', 4)
tennis_data.Temperature = tennis_data.Temperature.replace('Cool', 5)

tennis_data.Humidity = tennis_data.Humidity.replace('High', 6)
tennis_data.Humidity = tennis_data.Humidity.replace('Normal', 7)

tennis_data.Wind = tennis_data.Wind.replace('Weak', 8)
tennis_data.Wind = tennis_data.Wind.replace('Strong', 9)

tennis_data.PlayTennis = tennis_data.PlayTennis.replace('No', 10)
tennis_data.PlayTennis = tennis_data.PlayTennis.replace('Yes', 11)

tennis_data
```

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 데이터 전처리

>>

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	0	3	6	8	10
1	0	3	6	9	10
2	1	3	6	8	11
3	2	4	6	8	11
4	2	5	7	8	11
5	2	5	7	9	10
6	1	5	7	9	11
7	0	4	6	8	10
8	0	5	7	8	11
9	2	4	7	8	11
10	0	4	7	9	11
11	1	4	6	9	11
12	1	3	7	8	11
13	2	4	6	9	10

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 데이터 셋

```
X = np.array(pd.DataFrame(tennis_data, columns = ['Outlook', 'Temperature', 'Humidity', 'Wind']))
y = np.array(pd.DataFrame(tennis_data, columns = ['PlayTennis']))

X_train, X_test, y_train, y_test = train_test_split(X, y)

print('X_train :', X_train)
print('X_test :', X_test)
print('y_train :', y_train)
print('y_test :', y_test)
```

`

```
X_train : [[1 4 6 9]
 [2 4 6 8]
 [0 3 6 9]
 [0 4 7 9]
...

 [11]
 [11]
 [10]
 [10]]
y_test : [[11]
 [10]
 [11]
 [11]]
```

`

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 나이브 베이즈 모델 생성

```
gnb_clf = GaussianNB()
gnb_clf = gnb_clf.fit(X_train, y_train)

gnb_prediction = gnb_clf.predict(X_test)

print(gnb_prediction)
`


[10 10 11 10]

`
```

나이브 베이즈(Naive Bayes) 기법

● 예제를 이용한 Naive Bayes Python 코드 실습

◦ 성능 테스트

```
'''
Naive Bayes 모델의 predict함수를 사용해 X_test 데이터에 대한 예측값과 실제값 y_test를 비교해 모델의 성능을 평가하겠습니다.
성능 평가에 사용될 평가 요소는 confusion_matrix, classification_report, f1_score, accuracy_score입니다.성능 평가를 하기 위해 sklearn.metrics 패키지의 confusion_matrix, classification_report, f1_score, accuracy_score 모듈을 import합니다.
'''

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

'''
Confusion Matrix는 오차행렬을 나타냅니다. Confusion Matrix의 결과를 보면 2x2 행렬인 것을 알 수 있습니다.
Confusion Matrix의 y축은 실제값, x축은 예측값입니다.
'''

print('Confusion Matrix')
print(confusion_matrix(y_test, gnb_prediction))
`
```

```
Confusion Matrix
[[1 0]
 [3 0]]`
```

2. 분류 및 회귀

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 성능 테스트

'''
Classification Report는 분류에 대한 측정 항목을 보여주는 보고서를 나타냅니다.

Classification Report의 측정 항목으로는 클래스 별의 precision, recall, f1-score와 전체 데이터의 precision, recall, f1-score가 있습니다.
'''

```
print('Classification Report')
print(classification_report(y_test, gnb_prediction))
```

`

```
Classification Report
              precision    recall  f1-score   support

          10       0.25      1.00      0.40         1
          11       0.00      0.00      0.00         3

    accuracy                           0.25         4
   macro avg       0.12      0.50      0.20         4
weighted avg       0.06      0.25      0.10         4
```

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 성능 테스트

실제값과 예측값에 f1-score함수를 사용해 구한 f-measure와 accuracy_score 함수를 사용해 구한 accuracy를 나타내보겠습니다.

'''
f1_score 함수에 파라미터로 실제값 y_test와 예측값 gnb_prediction을 넣고 average를 weighted로 설정합니다.
weighted는 클래스별로 가중치를 적용하는 역할을 합니다. 이렇게 3개의 파라미터를 넣고 f1_score를 구한 후
round 함수를 이용해 소수점 2번째 자리까지 표현한 값을 변수 fmeasure에 저장합니다.
'''

```
fmeasure = round(f1_score(y_test, gnb_prediction, average = 'weighted'), 2)
```

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 성능 테스트

'''
accuracy_score 함수에 파라미터로 실제값 y_test와 예측값 gnb_prediction을 넣고 normalize를 True로 설정합니다.
True는 정확도를 계산해서 출력해주는 역할을 합니다. False로 설정하게 되면 올바르게 분류된 데이터의 수를 출력합니다.
이렇게 3개의 파라미터를 넣고 accuracy를 구한 후 round 함수를 이용해 소수점 2번째 자리까지 표현한 값을 변수 accuracy에 저장합니다.
'''

```
accuracy = round(accuracy_score(y_test, gnb_prediction, normalize = True), 2)

# 컬럼이 Classifier, F-Measure, Accuracy인 데이터프레임을 변수 df_nbclf에 저장합니다.
df_nbclf = pd.DataFrame(columns=['Classifier', 'F-Measure', 'Accuracy'])
```

나이브 베이즈(Naive Bayes) 기법

- 예제를 이용한 Naive Bayes Python 코드 실습
  - 성능 테스트

```
'''
컬럼 Classifier에는 Naive Bayes로 저장하고, 데이터프레임 df_nbclf에 loc 함수를 사용해
컬럼에 맞게 fmeasure 데이터와 accuracy 데이터를 데이터프레임에 저장합니다.
'''
df_nbclf.loc[len(df_nbclf)] = ['Naive Bayes', fmeasure, accuracy]


# 저장한 데이터프레임을 출력합니다.
df_nbclf
```

`

	Classifier	F-Measure	Accuracy
0	Naive Bayes	0.1	0.25

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 로지스틱 회귀분석을 위해서는 우선 예측하고자 하는 목표변수 Y의 범주가 0,1 두 가 지만 있다고 할 때(이항 로지스틱 회귀모델을 가정) 목표변수 Y의 범주가 1이 될 확률 p(Y=1) = P(Y) 로 표기하면, 이를 회귀식으로 다음과 같이 표현하는 것에서 시작한다. 아래 식에서 좌변은 흔히 오즈(odds)라고 불린다.

$$\frac{P(Y)}{1-P(Y)} = exp(\beta_0 + \beta_1 X)$$

- 위 식의 좌변은 확률들의 비율이고 우변은 지수함수의 형태이므로, (0, ∞)의 범위를 가지고 있음을 알 수 있다. 이에 좌변 및 우변 모두 (-∞,∞)값의 범위를 가지게 하기 위해 양변에 log 함수를 취한다.

$$\log\left(\frac{P(Y)}{1-p(Y)}\right) = \beta_0 + \beta_1 X$$

- 식을 자세히 보면, 우변의 $\beta_0 + \beta_1 X$ 는 선형모델이므로 (-∞,∞) 범위의 값을 가지 고, 좌변 역시 (-∞,∞) 범위의 값을 가짐을 알 수 있다. 좌변의 $\log\left(\frac{P(Y)}{1-p(Y)}\right)$ 형태 를 로짓(logit) 함수라고도 부른다.

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 혹은 위 식의 양변에 다시 지수 exp를 취하고, P(Y)에 대하여 식을 정리하면 다음과 같이 표현할 수도 있다.

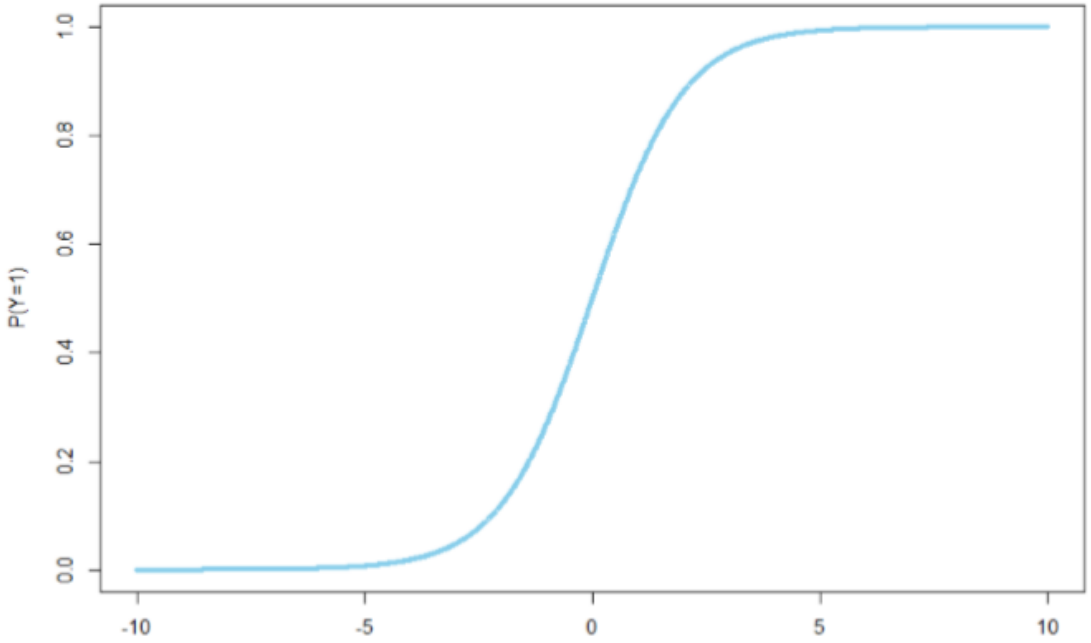
$$P(Y=1) = \frac{e^{(\beta_0 + \beta_1 X)}}{1 + e^{(\beta_0 + \beta_1 X)}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

- 로지스틱 회귀분석은 주어진 훈련 데이터에서 목표변수 Y가 범주 값 1을 가질 확률 P(Y=1) = P(Y)를 위의 식의 로지스틱 함수를 이용하여 모형을 수립하고 모수 β_0, β_1 들을 추정하는 알고리즘이라고 할 수 있다. 이러한 모수 β_0, β_1 추정에는 일 반적으로 최대우도추정법(Maximum Likelihood Estimation)을 사용하고, 이는 수식을 변형하는 것 등의 해석학적 방법으로 직접 계산이 어려우므로 일정 초기값을 부여한 뒤 이를 반복적으로 계산하여 값을 조정해 가는 수치 계산적 방법으로 추정하게 된다.
  - 확률(Probability) : 고정된 확률분포에서 어떠한 관측값이 나타나는지에 대한 확률
  - 우도(가능도, Likelihood) : 고정된 관측값이 어떠한 확률분포에서 어느정도의 확률로 나타나는지에 대한 확률

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 그림은 이러한 로지스틱 함수의 그래프 모양을 나타낸다. 그래프 형태에서 알 수 있듯 X축은 -∞부터 ∞까지의 값을 가지고 있으며 Y축의 경우 목표변수 Y의 발생확률이므로, 0에서 1 사이의 범위 값을 가지며 그 결과 함수인 P(Y=1)는 X값의 증가에 따라 0에서 1로 'S자 곡선'의 형태를 보인다는 것을 알 수 있다.



2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 로지스틱 회귀분석의 장/단점

장 점	단 점
✓ 선형통계모형의 이론에 기반한 정교하고 체계적인 모수 추정이 가능하다. ✓ 확률모형이므로, 목표변수의 범주 확률값을 추정할 수 있다. ✓ 추정된 모형의 계수에 대한 해석이 가능하며, 독립변수들의 유의성 및 영향력 등 결과 분석 시 유용한 해석이 가능하다	✓ 데이터 세트의 차원이 매우 많을 때 모형의 추정 정확도가 다른 머신러닝 기법에 비해 좋지 않다. ✓ 추정 방법상 x값이 매우 커지거나 작아지면 확률값 이 1(혹은 0)에 매우 가까워져서 수치계산 정확도가 떨어지게 되며, 반복 계산 시 오버 피팅이 빈번하게 발생한다. ✓ 복잡한 비선형적 분류가 필요한 경우에는 분류 정확도가 좋지 않다

로지스틱 함수 (logistic function)

- **로지스틱 회귀분석의 활용 분야**
  - 로지스틱 회귀분석은 일반적으로 금융권의 신용평가모형이나 이탈예측 모형 등에서 전통적으로 많이 활용되어 왔다. 그 외에도 구매예측이나 마케팅 반응 예측 등 수치 데이터를 활용한 분류 예측 문제에 많이 활용되고 있다.

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 유방암 특징을 측정한 데이터에 로지스틱 회귀 분석을 수행하여 유방암 발생을 예측

항목	내용
목표	로지스틱 회귀 분석을 이용해 유방암 진단에 영향을 미치는 특정 데이터를 분석하고, 유방암 여부를 예측하는 모델을 생성합니다.
핵심 개념	로지스틱 회귀, 시그모이드 함수, 성능 평가 지표 (혼동 행렬, 정확도, 정밀도, 재현율, F1 스코어), ROC 곡선 및 AUC 점수.
데이터 준비	유방암 진단 데이터: 사이킷런에서 제공하는 내장 데이터셋.
데이터 탐색	1. 데이터셋 설명 확인 (b_cancer.DESCR) 2. 지정된 X(피처)와 y(타깃)를 결합 3. 로지스틱 회귀를 위해 X 피처를 정규화: b_cancer_scaled = scaler.fit_transform(b_cancer.data)
분석 모델 구축	사이킷런의 로지스틱 회귀 모델 구축
결과 분석	성능 평가 지표 계산: confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 데이터 준비하기

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer

b_cancer = load_breast_cancer()

print(b_cancer.DESCR)
>>
```

…

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 데이터 준비하기

```
b_cancer_df = pd.DataFrame(b_cancer.data, columns = b_cancer.feature_names)
b_cancer_df['diagnosis']= b_cancer.target
b_cancer_df.head()
>>
```

mean radius	mean texture	mean perimeter	mean area	mean smoothness		mean compactness		mean concavity	mean
concave points	mean symmetry	mean fractal dimension		...	worst texture	worst perimeter	worst area	worst smoothness	
	worst compactness		worst concavity	worst concave points		worst symmetry	worst fractal dimension		
	diagnosis								
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
	0.4601	0.11890	0						
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
	0.2750	0.08902	0						
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
	0.3613	0.08758	0						
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575
	0.6638	0.17300	0						
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809
	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625
	0.2364	0.07678	0						

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 데이터 준비 및 탐색 : 데이터셋의 크기와 독립 변수 X가 되는 피처에 대한 정보를 확인. b_cancer_df.shape를 사용하여 데이터셋의 행의 개수(데이터 샘플 개수)와 열의 개수(변수 개수)를 확인 행의 개수가 569이므로 데이터 샘플이 569개, 열의 개수가 31이므로 변수가 31개 있음

```
print('유방암 진단 데이터셋 크기: ', b_cancer_df.shape)
>>
유방암 진단 데이터셋 크기:  (569, 31)
```

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 데이터 준비 및 탐색 : b_cancer_df에 대한 정보를 확인
    b_cancer_df.info( ) / 30개의 피처(독립 변수 X) 이름과 1개의 종속 변수 이름을 확인 가능 diagnosis는 악성이면 1, 양성이면 0의 값이므로 유방암 여부에 대한 이진 분류의 class로 사용할 종속 변수가 됨

```
b_cancer_df.info()
```

```
>>
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   mean radius            569 non-null    float64
 1   mean texture           569 non-null    float64
 2   mean perimeter         569 non-null    float64
 3   mean area              569 non-null    float64
 4   mean smoothness        569 non-null    float64
 ...
26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
 30  diagnosis               569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 데이터 준비 및 탐색 : 로지스틱 회귀 분석에 피처로 사용할 데이터를 평균이 0, 분산이 1이 되는 정규 분포 형태로 맞춤

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
b_cancer_scaled = scaler.fit_transform(b_cancer.data)

print(b_cancer.data[0])
>>
<[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
```

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 데이터 준비 및 탐색 : 로지스틱 회귀 분석에 피처로 사용할 데이터를 평균이 0, 분산이 1이 되는 정규 분포 형태로 맞춤
  - 사이킷런의 전처리 패키지에 있는 정규 분포 스케일러를 임포트하고 사용할 객체scaler를 생성
  - 피처로 사용할 데이터b_cancer.data에 대해 정규 분포 스케일링을 수행 scaler.fit_ transform( )하여 b_cancer_scaled에 저장
  - 정규 분포 스케일링 후에 값이 조정된 것을 확인

```
print(b_cancer_scaled[0])
>>
[ 1.09706398 -2.07333501  1.26993369  0.9843749   1.56846633  3.28351467
  2.65287398  2.53247522  2.21751501  2.25574689  2.48973393 -0.56526506
  2.83303087  2.48757756 -0.21400165  1.31686157  0.72402616  0.66081994
  1.14875667  0.90708308  1.88668963 -1.35929347  2.30360062  2.00123749
  1.30768627  2.61666502  2.10952635  2.29607613  2.75062224  1.93701461]
```

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 분석 모델 구축 및 결과 분석

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

#X, Y 설정하기
Y = b_cancer_df['diagnosis']
X = b_cancer_scaled

#훈련용 데이터와 평가용 데이터 분할하기
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0)

#로지스틱 회귀 분석: (1) 모델 생성
lr_b_cancer = LogisticRegression()

#로지스틱 회귀 분석: (2) 모델 훈련
lr_b_cancer.fit(X_train, Y_train)

#로지스틱 회귀 분석: (3) 평가 데이터에 대한 예측 수행 -> 예측 결과 Y_predict 구하기
Y_predict = lr_b_cancer.predict(X_test)
```

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 분석 모델 구축 및 결과 분석

```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score

#오차 행렬
confusion_matrix(Y_test, Y_predict)
>>
array([[ 60,   3],
       [  1, 107]], dtype=int64)
```

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 분석 모델 구축 및 결과 분석

```
acccuracy = accuracy_score(Y_test, Y_predict)
precision = precision_score(Y_test, Y_predict)
recall = recall_score(Y_test, Y_predict)
f1 = f1_score(Y_test, Y_predict)
roc_auc = roc_auc_score(Y_test, Y_predict)

print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}, F1:
{3:.3f}'.format(acccuracy,precision,recall,f1))
```

```
>>
정확도: 0.977, 정밀도: 0.973, 재현율: 0.991, F1: 0.982
```

2. 분류 및 회귀

로지스틱 함수 (logistic function)

- 특징 데이터로 유방암 진단하기
  - 분석 모델 구축 및 결과 분석 : 평가를 위해 7:3으로 분할한 171개의 test 데이터에 대해 이진 분류의 성능 평가 기본이 되는 오차 행렬을 구함. 실행 결과를 보면 TN이 60개, FP가 3개, FN이 1개, TP가 107개인 오차 행렬이 구해짐
  - 성능 평가 지표인 정확도, 정밀도, 재현율, F1 스코어, ROC-AUC 스코어를 구함

```
print('ROC_AUC: {0:.3f}'.format(roc_auc))
```

```
>>
ROC_AUC: 0.972
```

『 4과목 :』
데이터 분석과 인사이트 도출

- 통계와 확률 / 통계분석
- 머신러닝 기반 데이터 분석-지도 : 트리를 이용한 데이터 분석
- 머신러닝 기반 데이터 분석-비지도
- 기타 데이터 마이닝

- 『4과목』 Self 점검

Syallabus

학습목표

- 이 워크샵에서는 의사결정트리 기법, 앙상블 학습Ensemble Learning / 배깅(bagging) , 랜덤 포레스트(Random Forest) / Boosting에 대해 알 수 있습니다.

눈높이 체크

- 의사결정트리 기법, 앙상블 학습Ensemble Learning / 배깅(bagging) , 랜덤 포레스트(Random Forest) / Boosting을 알고 계시나요?

1. 의사결정트리 기법

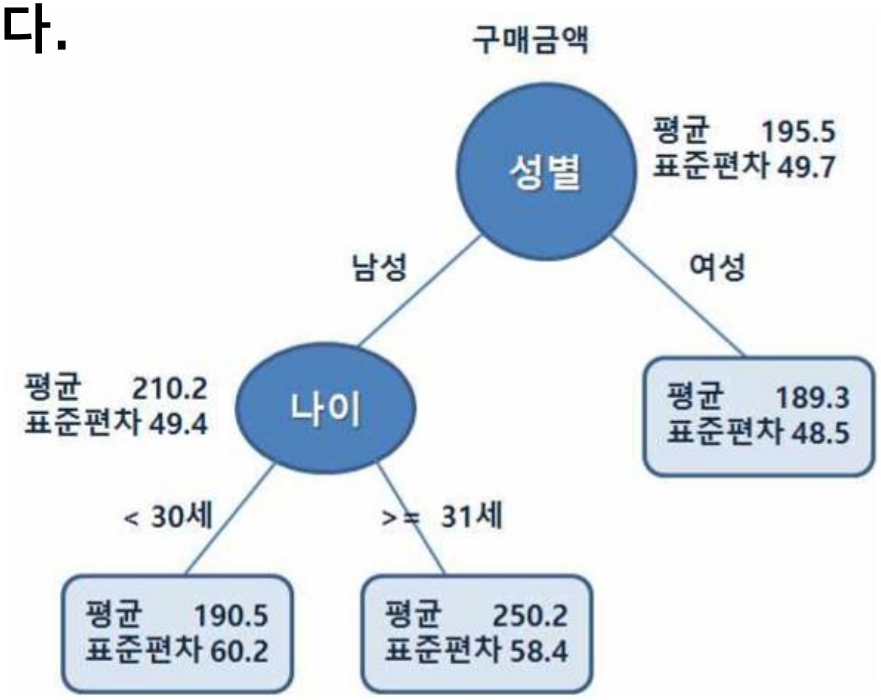
개념

- 의사결정트리는 분류 목적의 머신러닝 기법이지만, 수치예측의 목적으로도 사용될 수 있습니다. 어떤 경우에는 분류 목적의 의사결정 트리를 분류나무(Classification Tree)라고 부르고, 수치예측 목적의 의사결정트리를 회귀나무(Regression Tree)라고 구별해서 부르기도 합니다.
- 분류 목적의 의사결정트리가 노드 분리를 위해 카이제곱 통계량, 지니계수, 엔트로피 지수 등의 불순도(impurity)를 측정해서 트리를 구성하는 것과는 달리, 수치예측 목적의 의사결정트리는 목표변수의 평균과 표준편차 혹은 평균과 절대 편차 같은 통계치를 이용하여 불순도를 이용하여 마디가 분리됩니다.

1. 의사결정트리 기법

개념

- 그림 에서 구매금액을 목표변수로 하고 이에 영향을 미치는 설명변수들로 마디를 분리한다고 했을 때 첫 번째 분리변수로 성별이 사용된 것을 알 수 있는데, 남성의 구매금액 평균은 210만 원이고, 여성의 구매금액 평균은 189만 원이 되어 성별에 따라 구매금액의 높고 낮음이 잘 예측될 수 있기 때문입니다.



수치예측 목적의 의사결정 트리 예시

1. 의사결정트리 기법

의사결정트리 분리기준

- 수치예측 목적의 의사결정트리에서는 F 통계량의 p 값이나 분산(혹은 표준편차)의 감소량 등을 통해 가지를 분리하게 됩니다.
  - F 통계량의 p 값
  - p-값이 가장 작은 설명변수와 그때의 최적분리에 의해서 자식 노드가 형성됩니다. F통계량은 다음과 같이 계산됩니다.

$$F = (n_0 - 1)*(SST/SSE),$$
$$SST = n_1*(\overline{y_1} - \overline{y_0})^2 + n_2*(\overline{y_2} - \overline{y_0})^2,$$
$$SSE = SSE_L + SSE_R,$$
$$SSE_L = (n_1 - 1)*s_1^2 \quad , \quad SSE_R = (n_2 - 1)*s_2^2$$

  - 여기서, n0는 부모마디 데이터 세트 표본 크기, n1은 왼쪽마디 데이터 세트 표본 크 기, n2는 오른쪽마디 데이터 세트 표본 크기이며, ȳ0는 부모마디 목표변수 평균값, ȳ1는 왼쪽마디의 목표변수 평균값, ȳ2는 오른쪽 마디의 목표변수 평균값이고, s1는 왼쪽 마디의 목표변수 표준편차, s2는 오른쪽 마디의 목표변수 표준편차를 의미한 다. 즉, 특정변수로 목표변수의 평균값을 두 개의 노드로 분리시켰을 때 부모 노드 목표변수 평균값과의 제곱 합 감소량이 가장 커지는 F 값을 찾아서 F 값의 p 값이 가장 작은 경우로 자식 노드를 만들어 나가는 것이다.

1. 의사결정트리 기법

의사결정트리 분리기준

◦ 표준편차 혹은 분산의 감소량

- 표준편차 감소량(SDR: Standard Deviation Reduction) 혹은 분산의 감소량 (Variance Reduction)은 예측 오차를 최소화하는 것과 동일한 기준으로서, 목표변수를 분리하였을 때 표준편차 혹은 분산의 감소량을 최대화하는 기준의 최적분리에 의해서 자식 노드가 형성 됩니다. 표준편차 혹은 분산 감소량은 다음과 같이 측정됩니다.

$$VR = s_0^2 - (n_1/n_0)*s_1^2 - (n_2/n_0)*s_2^2 , \qquad (3.15)$$
$$SDR = s_0 - (n_1/n_0)*s_1 - (n_2/n_0)*s_2$$

$$SDR = SD(N) - \sum_i \frac{|N_i|}{|N|} * SD(N_i) \qquad (3.16)$$

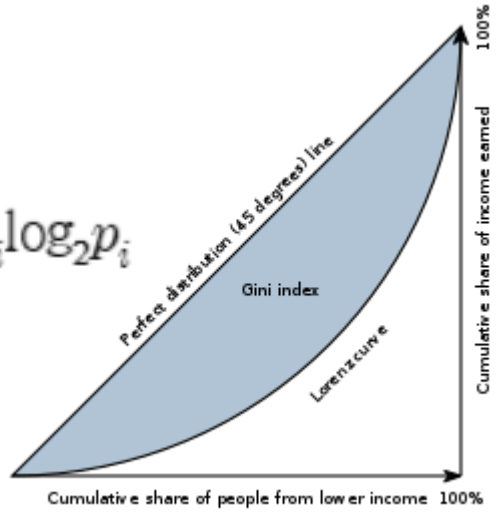
- 위 식 (3.15)에서와 같이 분산 감소량(VR)과 표준편차 감소량(SDR)은 본질적으로 동일한 식이며 식 (3.16)은 식 (3.15)을 일반화하여 표현한 것이다. 여기서 SD(N)은 부모 노드의 목표변수 값의 표준 편차를 의미합니다. 결국 위 공식은 부모 노드 목표 변수값의 표준편차를 노드 분리 후의 표준편차 값을 뺄셈하여 변화량을 측정하되, 노드가 분리된 후의 해당 노드의 데이터 표본 개수만큼의 가중치를 적용하여 표준편차(혹은 분산)의 감소량을 측정합니다. 이렇게 해서 이런 감소량이 가장 최대화되는 기준의 최적분리에 의해서 트리 분할이 이루어지게 됩니다.

1. 의사결정트리 기법

의사결정트리 분리기준

- 지니 계수(- 係數, 영어: Gini coefficient, 이탈리아어: coefficiente di Gini)는 경제적 불평등(소득 불균형)을 계수화 한 것이다. 오늘날 가장 널리 사용되는, 불평등의 정도를 나타내는 통계학적 지수로, 이탈리아의 통계학자인 코라도 지니(Corrado Gini)가 1912년 발표한 논문 "Variabilità e mutabilità"에 처음 소개되었다.
- 엔트로피 지수 (Entropy)

$$Gini\ Coff. = 1 - \sum_{i=1}^{K} p_i^2 \quad , \quad Entropy\ Coff. = -\sum_{i=1}^{K} p_i \log_2 p_i$$



1. 의사결정트리 기법

의사결정트리기법의 장/단점

장 점	단 점
✓ 자동으로 특성(Feature)을 선택해주므로, 많은 변수를 가진 수치예측 문제에 사용이 가능하다. ✓ 결측치 및 이상치가 있는 데이터에도 효과적 으로 처리가능. ✓ 다중공선성의 문제나 선형성 가정, 정규성 가정 등이 필요 없다. ✓ 두 개 이상의 변수가 결합한 비선형적 상호작용효과를 해석하기가 용이하다. ✓ 수학적 지식이 없는 사람도 모형의 결과 이해가 쉽고, 어떤 입력변수가 목표변수를 설명하는데 영향력이 높은지를 알 수 있습니다.	✓ 연속형 입력변수를 비연속적인 값으로 취급하므로, 분리의 경계점 근방에서 예측 오류 가능성 있음. ✓ 선형 또는 주효과 모형과 같은 해석이 불가능하므 로 모형식을 수립해야 하는 경우 적용이 어렵다. ✓ 훈련데이터에 대한 약간의 변경이 발생시 트리분류 결정 논리에 큰 변화를 가져온다. (특정 데이터 변화에 분석결과 변화가 민감함) ✓ 모델이 쉽게 과적합화되거나 과소적합될 수 있습니다. ✓ 트리가 너무 커질 경우 패턴 이해하기가 쉽지 않다.

1. 의사결정트리 기법

의사결정트리기법의 장/단점

- 일반적으로 현업 실무 환경에서 수치예측목적의 의사결정트리는 분류 목적의 의사결정트리만큼 빈번하게 사용되지는 않는 경향이 있습니다. 그 이유는 목표변수가 연속형이면 예측력이 다소 떨어지는 경향이 있으며, 자료에 약간의 변화가 생겨도 민감하게 반응하는 경향이 있기 때문이다. 분류 목적의 경우에는 목표변수의 범주 속성별로 빈도가 높은 쪽으로 결과 판정이 일어나기 때문에 예측력이 다소 떨어져도 범주 라벨 예측에는 그다지 문제가 되지 않지만, 수치예측의 경우에는 결과값 자체를 정교하게 예측할 필요가 있기 때문에 예측력이 더 중요해지는 경향이 있습니다. 그렇지만 의사결정트리기법의 특성자체가 방대한 변수들이 있을 때 중요한 변수를 판별해 내거나, 규칙을 도출하는 데에 있어 해석이 직관적이기 때문에 경제 및 기업의 데이터 분석 업무에서 보다 정교한 분석 모형 수립 이전의 사전 탐색분석단계나 규칙도출 등의 목적으로 많이 활용되고 있습니다.

1. 의사결정트리 기법

타이타닉호 생존자 예측

```
import pandas as pd
df =  pd.read_csv('datasets/titanic_sns.csv')
df.head()

>>
```

	survived	pclass	Gender	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

1. 의사결정트리 기법

타이타닉호 생존자 예측

```
feature_names = ["pclass", "age", "Gender"]
dfX = df[feature_names].copy()
dfy = df["survived"].copy()
dfX.tail()

>>
```

	pclass	age	Gender
886	2	27.0	male
887	1	19.0	female
888	3	NaN	female
889	1	26.0	male
890	3	32.0	male

1. 의사결정트리 기법

타이타닉호 생존자 예측

```
dfX["age"].fillna(dfX["age"].mean(), inplace=True)
dfX.tail()
```

>>

	pclass	age	Gender
886	2	27.000000	1
887	1	19.000000	0
888	3	29.699118	0
889	1	26.000000	1
890	3	32.000000	1

1. 의사결정트리 기법

타이타닉호 생존자 예측

```
dfX["age"].fillna(dfX["age"].mean(), inplace=True)
dfX.tail()
```

>>

	pclass	age	Gender
886	2	27.000000	1
887	1	19.000000	0
888	3	29.699118	0
889	1	26.000000	1
890	3	32.000000	1

타이타닉호 생존자 예측

```
from sklearn.preprocessing import LabelBinarizer
import pandas as pd

dfX2 = pd.DataFrame(LabelBinarizer().fit_transform(dfX["pclass"]),
                    columns=['c1', 'c2', 'c3'], index=dfX.index)
dfX = pd.concat([dfX, dfX2], axis=1)
del(dfX["pclass"])
dfX.tail()

>>
```

	age	Gender	c1	c2	c3	
886	27.000000		1	0	1	0
887	19.000000		0	1	0	0
888	29.699118		0	0	0	1
889	26.000000		1	1	0	0
890	32.000000		1	0	0	1

2. Ensemble Learning

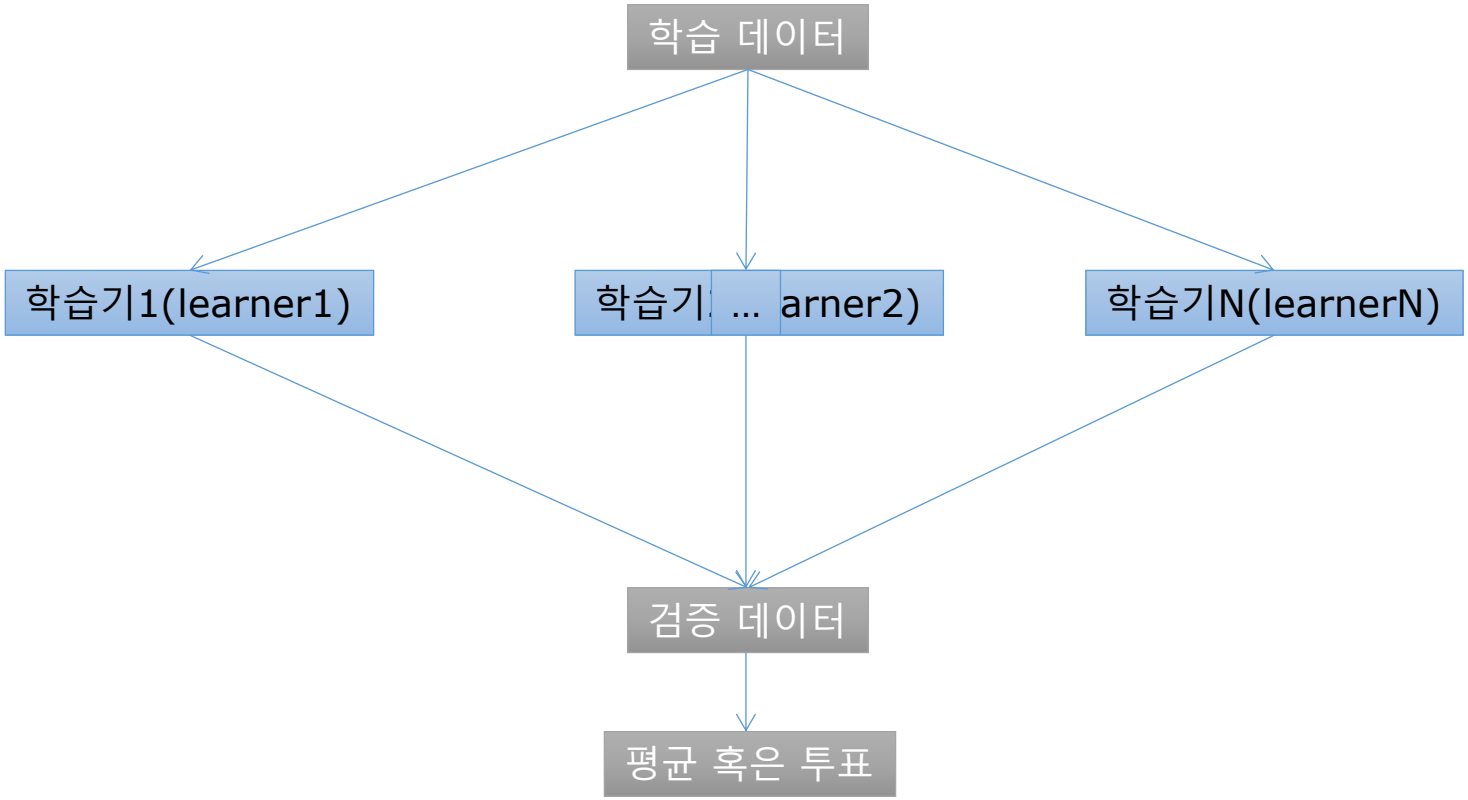
개념

- 하나의 데이터를 여러개의 분류기를 통해 다수의 학습 모델을 만들어 학습시키고 학습 결과를 결합함으로써 과적합을 방지하고 정확도를 높이는 학습 기법입니다.
- 유형
  - 보팅: 앙상블 학습의 기본, 하위 모든 기법들이 보팅 사용
  - 배깅: 하나의 데이터를 여러개로 나누어 학습하는 앙상블 학습법
    - 보팅을 하나의 고유한 학습법으로 보고 배깅과 구분하자면, 일반적으로 보팅은 하나의 데이터에 여러 알고리즘 적용, 배깅은 여러개로 나누어진 데이터에 하나의 알고리즘을 적용하는 것으로 구분
    - 하지만 여러개로 나누어진 데이터를 이용하는 배깅에서도, 최종 예측값을 선택하는 행위는 '보팅'이라 함
  - 부스팅: 병렬로 수행되는 배깅과 달리, 각 결과값을 이용하여 순차적으로 결합
  - 랜덤 포레스트: 배깅 + 의사결정 나무

2. Ensemble Learning

배깅(Bagging)

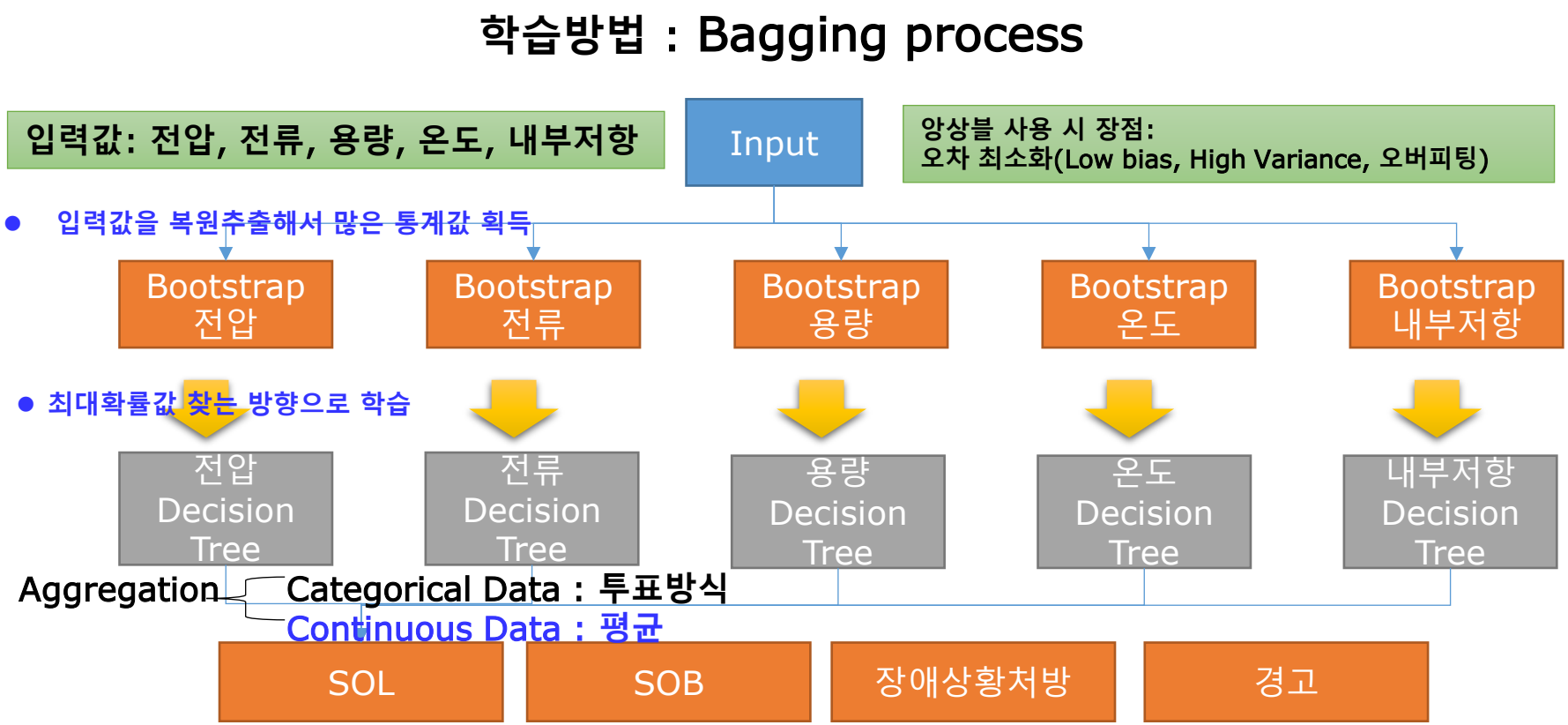
- 배깅(Bagging)은 Bootstrap Aggregating의 줄임말로, 부트스트래핑을 이용한 앙상블 학습법
- 부트스트래핑: 학습 데이터셋에서 중복을 허용하여 랜덤하게 추출하는 방식(aka. 리샘플링)
- 페이스팅: 학습 데이터셋에서 중복 없이 랜덤하게 추출하는 방식



2. Ensemble Learning

배깅(Bagging)

- 배깅(Bagging) 활용 예



2. Ensemble Learning

배깅(Bagging)

- 배깅(bagging) 해보기-집 값 예측

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

data=pd.read_csv("datasets/kc-house-data.csv")
data.head()
```

>>

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long
	sqft_living15	sqft_lot15							
0	7129300520	20141013T000000		221900.0	3	1.00	1180	5650	1.0
	0	0	...	7	1180	0	1955	0	98178
	47.5112	-122.257	1340	5650					
1	6414100192	20141209T000000		538000.0	3	2.25	2570	7242	2.0
	0	0	...	7	2170	400	1951	1991	98125
	47.7210	-122.319	1690	7639					
2	5631500400	20150225T000000		180000.0	2	1.00	770	10000	1.0
	0	0	...	6	770	0	1933	0	98028
	47.7379	-122.233	2720	8062					
3	2487200875	20141209T000000		604000.0	4	3.00	1960	5000	1.0
	0	0	...	7	1050	910	1965	0	98136
	47.5208	-122.393	1360	5000					
4	1954400510	20150218T000000		510000.0	3	2.00	1680	8080	1.0
	0	0	...	8	1680	0	1987	0	98074
	47.6168	-122.045	1800	7503					

5 rows × 21 columns

2. Ensemble Learning

배깅(Bagging)

- 배깅(bagging) 해보기-집 값 예측

```
nCar=data.shape[0]
nVar=data.shape[1]
print(nCar, nVar)
>>

21613 21

# 필요 없는 데이터 변수 제거
data=data.drop(['id','date','zipcode','lat','long'],axis=1)

# 특성 초기화(data=독립변수, 설명변수, target=종속변수)
feature_columns=list(data.columns.difference(['price']))
X=data[feature_columns]
y=data['price']

# 학습 데이터 셋, 시험 데이터 셋 구분
train_x,test_x,train_y,test_y=train_test_split(X,y,test_size=0.3,random_state=42)
print(train_x.shape,test_x.shape,train_y.shape,test_y.shape)
```

2. Ensemble Learning

배깅(Bagging)

- 배깅(bagging) 해보기-집 값 예측

```
# 라이브러리 임포트
from sklearn.linear_model import LinearRegression
import math
from sklearn.metrics import mean_squared_error
# 선형회귀모델생성
regression_model=LinearRegression()
# 훈련
linear_model1=regression_model.fit(train_x, train_y)
# 예측
predict1=linear_model1.predict(test_x)
# 결과 인쇄
print('RMSE: {}'.format(math.sqrt(mean_squared_error(predict1, test_y))))
>>
RMSE: 223893.6056181597
```

2. Ensemble Learning

Random Forest

- 다양한 앙상블 기법중에서 랜덤 포레스트(random forest)와 그래티언트 부스팅(gradient boosting)결정 트리는 둘 다 모델을 구성하는 기본 요소로 결정 트리를 사용합니다.
- 결정 트리의 주요 단점은 훈련 데이터에 과대적합되는 경향이 있다는 것이었다.
- 랜덤 포레스트는 이 문제를 회피할 수 있는 방법이다.랜덤 포레스트는기본적으로 여러 결정 트리의 묶음이라고 보면 됩니다. 각 트리는 비교적 예측을 잘 하구 있지만 일부에 과대 적합하다는 경향을 가지고 있음에 기초합니다.
- 서로 다른 방향으로 과대적합된 트리를 많이 만들면 그 결과를 평균냄으로써 과대적합된 양을 줄일 수 있습니다.이러한 전략은 구현하기 위해서는 결정 트리를 많이 만들어야 합니다. 랜덤 포레스트에서 트리를 랜덤하게 만드는 방법은 두가지가 있습니다.

1. 데이터를 무작위로 선택하기
2. 분할 테스트에서 특성을 무작위로 선택하기

2. Ensemble Learning

Random Forest

- **Iris 데이터를 이용해 간단한 랜덤포레스트 분류**

```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd

#loading the iris dataset
iris = load_iris()

#training data 설정
x_train = iris.data[:-30]
y_train = iris.target[:-30]
#test data 설정
x_test = iris.data[-30:] # test feature data
y_test = iris.target[-30:] # test target data

print(y_train)

print(y_test)
>>
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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 2 2]
```

Random Forest

- Iris 데이터를 이용해 간단한 랜덤포레스트 분류

```
rfc.fit(x_train, y_train)
#Test data를 입력해 target data를 예측
prediction = rfc.predict(x_test)
#예측 결과 precision과 실제 test data의 target 을 비교
print (prediction==y_test)

>>

[ True  True  True False  True  True False False  True False  True  True
  True False False  True  True  True False  True  True  True  True  True
  True  True  True  True  True  True]

#Random forest 정확도 츶정
rfc.score(x_test, y_test)


>>
0.8
```

2. Ensemble Learning

Random Forest

- 랜덤 포레스트 분류 성능 평가

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


print ("Accuracy is : ",accuracy_score(prediction, y_test))
print ("==================================================")
print (classification_report(prediction, y_test))
```

```
>>
Accuracy is :  0.8
==================================================
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         6
           2       0.80      1.00      0.89        24

    accuracy                           0.80        30
   macro avg       0.40      0.50      0.44        30
weighted avg       0.64      0.80      0.71        30
```

2. Ensemble Learning

Random Forest

- Random forest 성능 제고

```
clf = RandomForestClassifier(n_estimators=10)
clf.fit(X_train, Y_train)
prediction_1 = rfc.predict(X_test)
#print (prediction_1 == Y_test)
print ("Accuracy is : ",accuracy_score(prediction_1, Y_test))
print ("==========================================================")
print (classification_report(prediction_1, Y_test))
```

```
>>
Accuracy is :  0.8333333333333334
==========================================================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      0.64      0.78        14
           2       0.69      1.00      0.81        11

    accuracy                           0.83        30
   macro avg       0.90      0.88      0.87        30
weighted avg       0.89      0.83      0.83        30
```

2. Ensemble Learning

Random Forest

- Random forest 성능 제고

```
# Initialize the model
clf_2 = RandomForestClassifier(n_estimators=200, # Number of trees
                               max_features=4,    # Num features considered
                                 oob_score=True)    # Use OOB scoring*
clf_2.fit(X_train, Y_train)
prediction_2 = clf_2.predict(X_test)
print (prediction_2 == Y_test)
print ("Accuracy is : ",accuracy_score(prediction_2, Y_test))
print ("==================================================")
print (classification_report(prediction_2, Y_test))
```

```
>>
[ True  True  True False  True  True  True  True False  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True]
Accuracy is :  0.9333333333333333
==================================================
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       0.89      0.89      0.89         9
           2       0.94      0.94      0.94        16

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```

Random Forest

- Random forest 성능 제고

```
for feature, imp in zip(iris.feature_names, clf_2.feature_importances_):
    print(feature, imp)
```

```
>>
sepal length (cm) 0.008110566699516389
sepal width (cm) 0.011003693678771487
petal length (cm) 0.6164259475896038
petal width (cm) 0.36445979203210843
```

2. Ensemble Learning

Random Forest

- 특성 중요도 분석

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import mglearn
import numpy as np

cancer = load_breast_cancer()

# 훈련/테스트 세트로 나누기
X_train, X_test, y_train, y_test = train_test_split(cancer.data,cancer.target, random_state=0)
forest = RandomForestClassifier(n_estimators=100,random_state=0)
forest.fit(X_train,y_train)

print("훈련 세트 정확도 : {:.3f}".format(forest.score(X_train,y_train)))
print("테스트 세트 정확도 : {:.3f}".format(forest.score(X_test,y_test)))

# 특성 중요도
print("특성 중요도 : \n{}".format(forest.feature_importances_))

>>
```

2. Ensemble Learning

Random Forest

- **특성 중요도 분석**

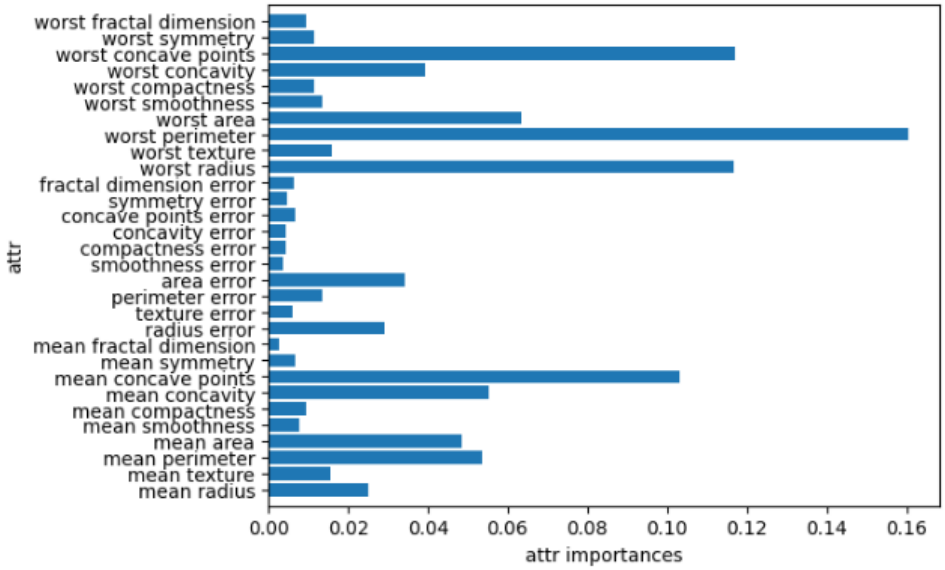
```
>>
훈련 세트 정확도 : 1.000
테스트 세트 정확도 : 0.972
특성 중요도 :
[0.02515433 0.01563844 0.05372655 0.04861645 0.00769078 0.00936994
 0.05539489 0.10305394 0.0065771  0.00282708 0.02921459 0.00607814
 0.01342868 0.03420174 0.00360641 0.00432096 0.00448775 0.00657502
 0.00460597 0.00627095 0.11657269 0.01603133 0.16027724 0.0634688
 0.01356448 0.01164113 0.03923725 0.11711756 0.01164259 0.00960721]
```

2. Ensemble Learning

Random Forest

- 특성 중요도 분석

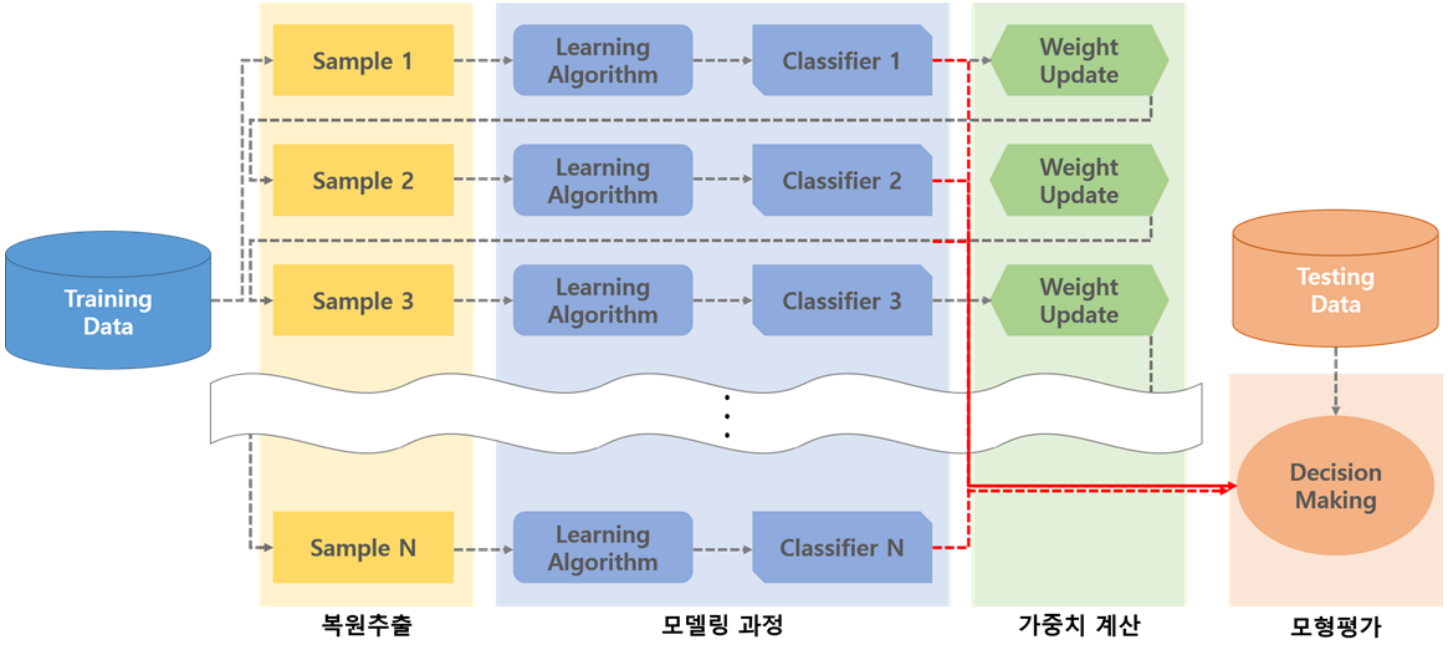
```
# 특성 중요도 시각화 하기

def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("attr importances")
    plt.ylabel("attr")
    plt.ylim(-1, n_features)

plt.show()

plot_feature_importances_cancer(forest)
>>
```



2. Ensemble Learning

Boosting

- 기존 학습 데이터에서 random sampling을 하고 1번 weak learner로 학습시킨다. 그 결과로 생긴 에러를 반영해 그 다음 데이터 샘플링과 2번 weak learner를 잡고 학습을 반복합니다. 이 과정을 N번 하면 iteration N번 돌린 부스팅 모델이 되는 것이다. 부스팅 계열 모델은 AdaBoost, Gradient Boost(GBM), XGBoost, LightGBM, CatBoost 등이 있습니다.



2. Ensemble Learning

Boosting

- AdaBoost 사용해 보기

```
# 1. 에이다 부스트
# 라이브러리 임포트
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
# 데이터 로드-아이리스
iris=datasets.load_iris()
# 특성 초기화
data=iris.data
target=iris.target
# 부스팅 관련 분류기 객체 생성
adaboost=AdaBoostClassifier(random_state=0)
# 훈련
rs_ada=adaboost.fit(data, target)
rs_ada.feature_importances_

>>
array([0.  , 0.  , 0.44, 0.56])
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 사용해 보기

```
# 라이브러리 임포트
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import datasets
# 데이터 로드-아이리스
iris=datasets.load_iris()
# 특성 초기화
data=iris.data
target=iris.target
# 부스팅 관련 분류기 객체 생성
gradientboost=GradientBoostingClassifier(random_state=0)
# 훈련
rs_gb=gradientboost.fit(data, target)
rs_gb.feature_importances_

>>
array([0.0053083 , 0.01347136, 0.30658561, 0.67463473])
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 사용해 보기

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import mglearn
import numpy as np

cancer = load_breast_cancer()

# 훈련/테스트 세트로 나누기
X_train, X_test, y_train, y_test = train_test_split(cancer.data,cancer.target, random_state=0)

gbrt = GradientBoostingClassifier(random_state=0)

gbrt.fit(X_train,y_train)

print("훈련 세트 정확도 : {:.3f}".format(gbrt.score(X_train,y_train)))
print("테스트 세트 정확도 : {:.3f}".format(gbrt.score(X_test,y_test)))
>>
훈련 세트 정확도 : 1.000
테스트 세트 정확도 : 0.965
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 사용해 보기

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()

# 훈련/테스트 세트로 나누기
X_train, X_test, y_train, y_test = train_test_split(cancer.data,cancer.target, random_state=0)

gbrt = GradientBoostingClassifier(random_state=0)

gbrt.fit(X_train,y_train)

print("훈련 세트 정확도 : {:.3f}".format(gbrt.score(X_train,y_train)))
print("테스트 세트 정확도 : {:.3f}".format(gbrt.score(X_test,y_test)))

)))
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 사용해 보기

```
# 훈련 세트 정확도 : 1.000
# 테스트 세트 정확도 : 0.958
# 훈련 세트의 정확도가 100%이므로 과대적합되었다.
# 과대적합을 막기위해 사전 가지치기를 합니다.

gbrt = GradientBoostingClassifier(random_state=0,max_depth=1)

gbrt.fit(X_train,y_train)

print("훈련 세트 정확도 : {:.3f}".format(gbrt.score(X_train,y_train)))
print("테스트 세트 정확도 : {:.3f}".format(gbrt.score(X_test,y_test)))
>>
훈련 세트 정확도 : 1.000
테스트 세트 정확도 : 0.965
훈련 세트 정확도 : 0.991
테스트 세트 정확도 : 0.972
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 사용해 보기

과대적합을 막기위해 학습률을 낮춘다

```
gbrt = GradientBoostingClassifier(random_state=0,learning_rate=0.01)

gbrt.fit(X_train,y_train)

print("훈련 세트 정확도 : {:.3f}".format(gbrt.score(X_train,y_train)))
print("테스트 세트 정확도 : {:.3f}".format(gbrt.score(X_test,y_test)))
```

>>

```
훈련 세트 정확도 : 0.988
테스트 세트 정확도 : 0.958
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 특성 중요도

```
# 특성 중요도 시각화 하기

def plot_feature_importances_cancer(model):

    n_features = cancer.data.shape[1]

    plt.barh(range(n_features), model.feature_importances_, align='center')

    plt.yticks(np.arange(n_features), cancer.feature_names)

    plt.xlabel("attr importances")

    plt.ylabel("attr")

    plt.ylim(-1, n_features)

plt.show()

plot_feature_importances_cancer(gbrt)
```

2. Ensemble Learning

Boosting

- **Gradient Boost(GBM) 특성 중요도**

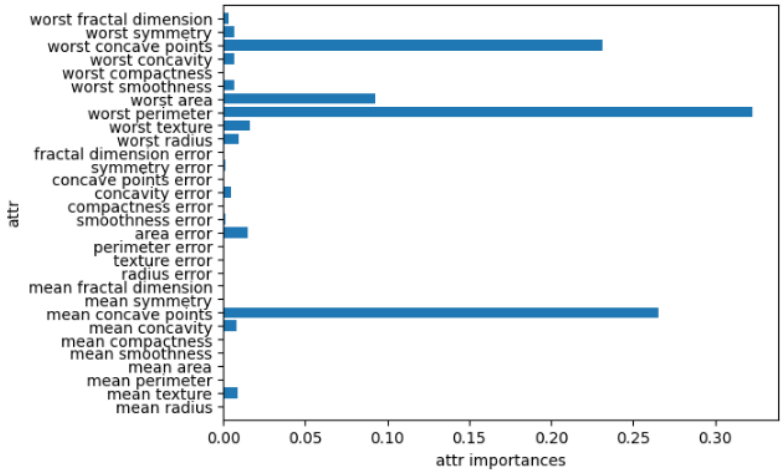
```
>>
훈련 세트 정확도 : 0.991
테스트 세트 정확도 : 0.972
특성 중요도 :
[0.00000000e+00 9.07529959e-03 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 8.32035766e-03 2.65686907e-01
 3.49588341e-04 0.00000000e+00 0.00000000e+00 3.00083378e-04
 0.00000000e+00 1.45838255e-02 1.05083243e-03 0.00000000e+00
 4.75469106e-03 0.00000000e+00 9.56890421e-04 0.00000000e+00
 9.23939383e-03 1.61819935e-02 3.22495788e-01 9.24249926e-02
 6.44041105e-03 0.00000000e+00 6.85101108e-03 2.31450675e-01
 6.41500790e-03 3.42225071e-03]
```

2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 특성 중요도

```
# 특성 중요도 시각화 하기

def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("attr importances")
    plt.ylabel("attr")
    plt.ylim(-1, n_features)

plt.show()

plot_feature_importances_cancer(gbrt)

>>
```

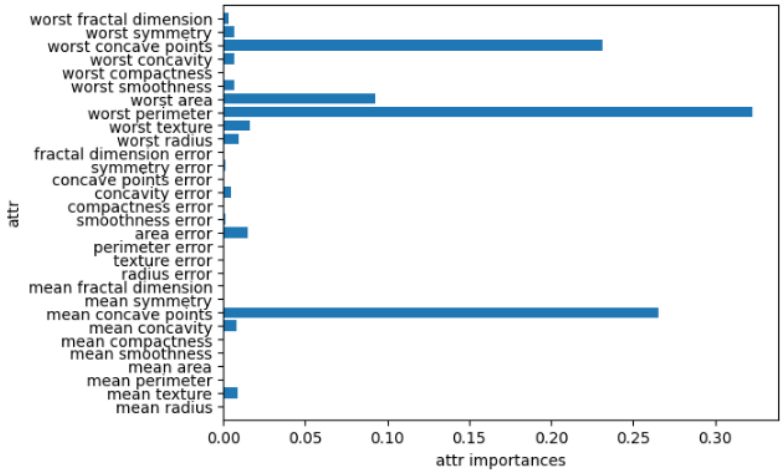


2. Ensemble Learning

Boosting

- Gradient Boost(GBM) 특성 중요도

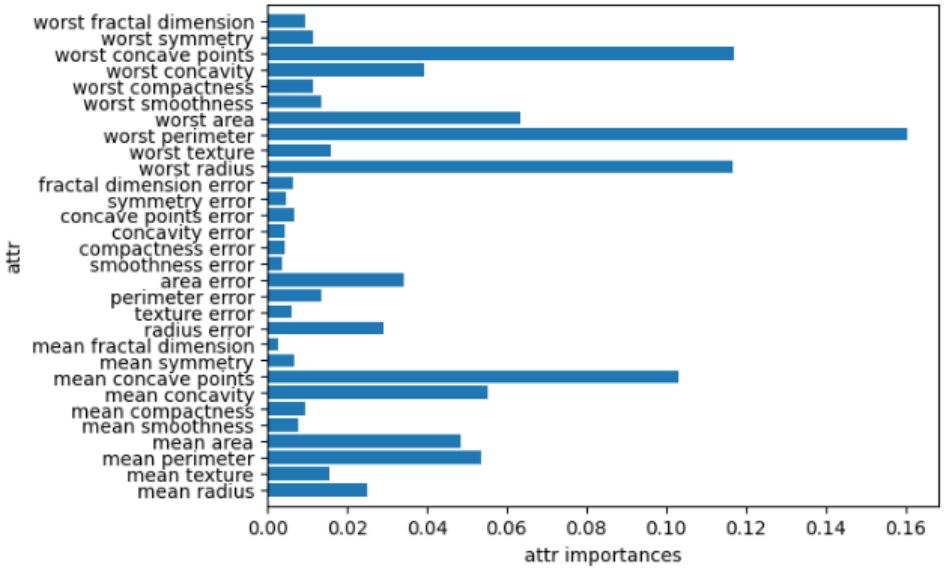
```
# 특성 중요도 시각화 하기

def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("attr importances")
    plt.ylabel("attr")
    plt.ylim(-1, n_features)

plt.show()

plot_feature_importances_cancer(gbrt)

>>
```

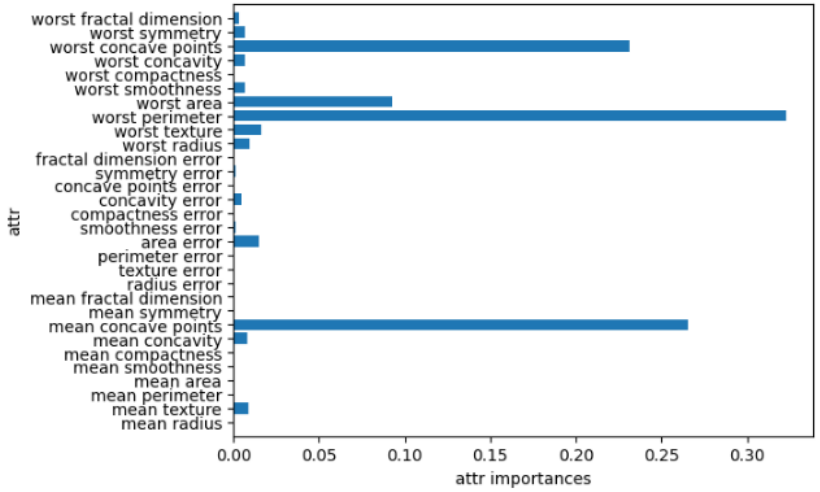


2. Ensemble Learning

Boosting

- Random Forest와 Gradient Boost(GBM) 특성 중요도 비교





『 4과목 :』
데이터 분석과 인사이트 도출

- 통계와 확률 / 통계분석
- 머신러닝 기반 데이터 분석-지도 : 수치 예측
- 머신러닝 기반 데이터 분석-비지도
- 기타 데이터 마이닝

- 『4과목』 Self 점검

Syallabus

학습목표

- 이 워크샵에서는 SVM(Support Vector Machine), 인공신경망 확인할 수 있다

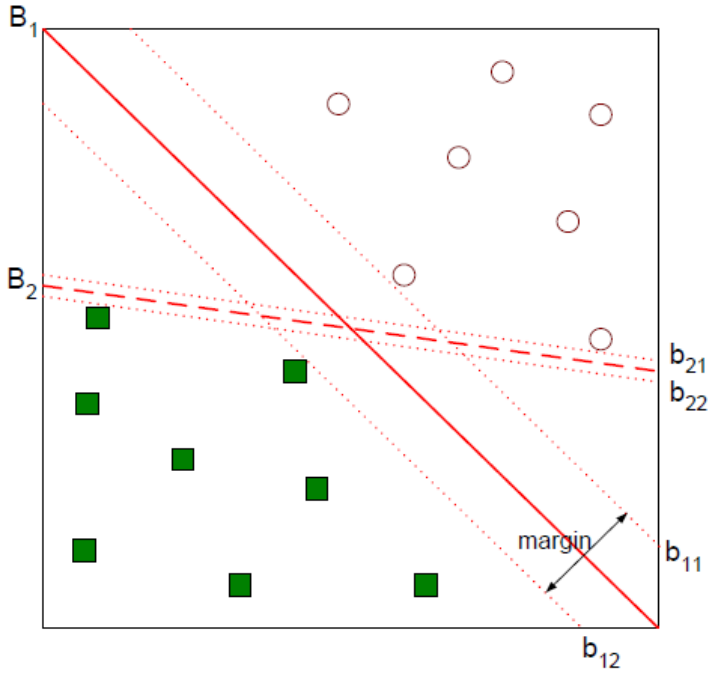
눈높이 체크

- SVM(Support Vector Machine), 인공신경망 을 알고 계시나요?

1. SVM(Support Vector Machine)

개념

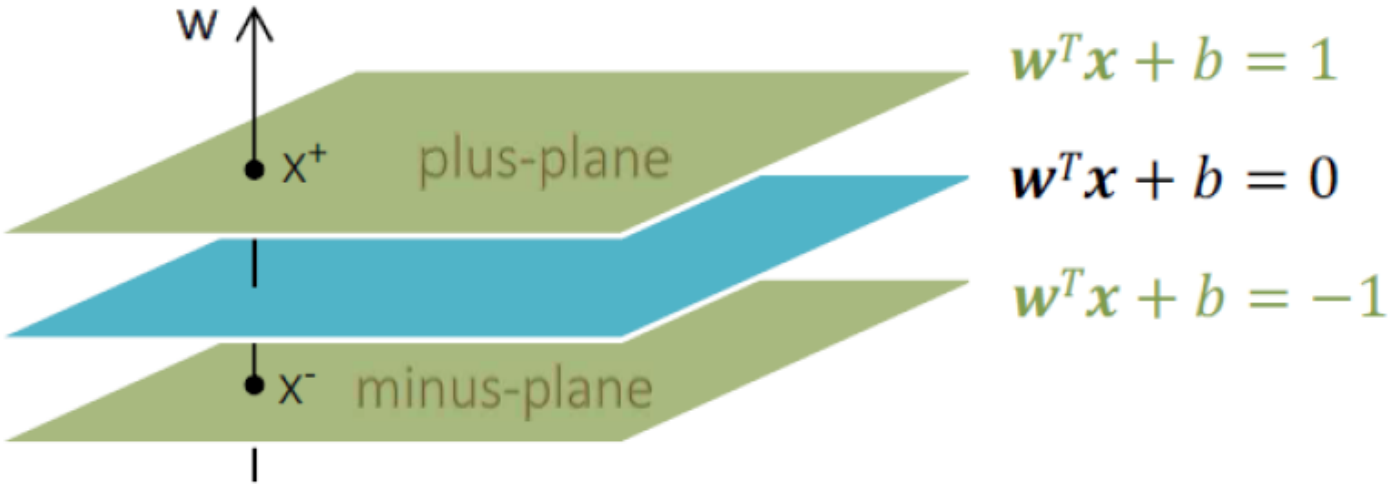
- 서포트 벡터(혹은 지지 벡터)머신은 서로 다른 분류에 속한 데이터 간에 간격(마진)이 최대가 되는 선(또는 초평면)을 찾아서 이를 기준으로 데이터를 분류하는 모델이다. 아래 그림에서 직선 B1과 B2 모두 두 클래스를 무난하게 분류하고 있음을 확인할 수 있다.



1. SVM(Support Vector Machine)

개념

- 좀 더 나은 분류경계면을 꼽으라면 B1이다. 위 그림에서 b12을 minus-plane, b11을 plus-plane, 이 둘 사이의 거리를 마진(margin)이라고 하며, SVM은 이 마진을 최대화하는 분류 경계면을 찾는 기법이다. 이를 도식적으로 나타내면 아래와 같다.



- 벡터 w는 이 경계면과 수직인 법선벡터 벡터 w는 이 경계면과 수직인 법선벡터

1. SVM(Support Vector Machine)

개념

- margin

$$
\begin{aligned}
Margin &= distance(x^+, x^-) \\
&= \|x^+ - x^-\|_2 \\
&= \|x^- + \lambda w - x^-\|_2 \\
&= \|\lambda w\|_2 \\
&= \lambda\sqrt{w^T w} \\
&= \frac{2}{w^T w}\sqrt{w^T w} \\
&= \frac{2}{\sqrt{w^T w}} \\
&= \frac{2}{\|w\|_2}
\end{aligned}
$$

벡터 x^+와 x^- 사이의 관계

$x^+ = x^- + \lambda w$

$$
\begin{aligned}
w^T x^+ + b &= 1 \\
w^T(x^- + \lambda w) + b &= 1 \\
w^T x^- + b + \lambda w^T w &= 1 \\
-1 + \lambda w^T w &= 1
\end{aligned}
$$

$$
\lambda = \frac{2}{w^T w}
$$

1. SVM(Support Vector Machine)

개념

- SVM의 목적은 마진을 최대화하는 경계면을 찾는 것
  - 계산상 편의를 위해 마진 절반을 제곱한 것에 역수를 취한 뒤 그 절반을 최소화 곧, 목적함수를 최소화

$$max \frac{2}{\|w\|_2} \rightarrow \min \frac{1}{2}\|w\|_2^2$$

  - 제약식

$$y_i(w^T x_i + b) \geq 1$$

1. SVM(Support Vector Machine)

개념

- 라그랑주 승수법(Lagrange multiplier method)
  - 라그랑주 승수법 (Lagrange multiplier method)은 프랑스의 수학자 조세프루이 라그랑주 (Joseph-Louis Lagrange)가 제약 조건이 있는 최적화 문제를 풀기 위해 고안한 방법이다. 라그랑주 승수법은 어떠한 문제의 최적점을 찾는 것이 아니라, 최적점이 되기 위한 조건을 찾는 방법이다. 즉, 최적해의 필요조건을 찾는 방법이다.

  - 목적식과 제약식을 라그랑지안 문제로 식을 다시 쓰면

$$\min L_p(w, b, \alpha_i) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1)$$

  - Lp 의 제약

$$\alpha i \geq 0, i = 1, \ldots, n$$

1. SVM(Support Vector Machine)

개념

- 라그랑주 승수 값이 0 즉, ai=0 이면 해당 데이터는 예측 모형, 즉 w 계산에 아무런 기여를 하지 않으므로 위 식을 실제로는 다음과 같다.

$$f(x) = a^{+}x^{T}x^{+} - a^{-}x^{T}x^{-} - w_0$$

- 여기에서 xTx+ 는 x와 x+사이의 (코사인)유사도, xTx− 는 x와 x−사이의 (코사인)유사도이므로 결국 두 서포트 벡터와의 유사도를 측정해서 값이 큰 쪽으로 판별하게 된다.

1. SVM(Support Vector Machine)

서포트 벡터 머신 기법의 장/단점

장 점	단 점
✓ 범주분류나 수치예측 문제에 모두 활용 가능하다. ✓ 노이즈 데이터에 영향을 크게 받지 않고, 과적합화가 잘 되지 않는다. ✓ 일반적으로 분류 문제에서 다른 알고리즘 보다 분류 성능이 높은 것으로 알려져 있으며, 특히 분류 경계가 복잡한 비선형 문제일 경우 타 기법대비 성능이 좋은 것으로 알려져 있	✓ 최적 분류를 위해 커널함수 및 매개변수 등에 대한 반복적인 조합 테스트가 필요하다. ✓ 입력 데이터의 양이나 변수가 많은 경우 훈련에 오랜 시간이 소요된다. ✓ 배경이 되는 이론 및 알고리즘 구현 시 타 기법에 비해 상대적으로 난해한 면이 있다. ✓ 결과 해석이나 이유 설명 등이 쉽지 않다.

1. SVM(Support Vector Machine)

Scikit-Learn의 서포트 벡터 머신

- Scikit-Learn의 svm 서브패키지는 서포트 벡터 머신 모형인 SVC (Support Vector Classifier) 클래스를 제공한다.
  - 사이킷런 SVM 주요 파라미터

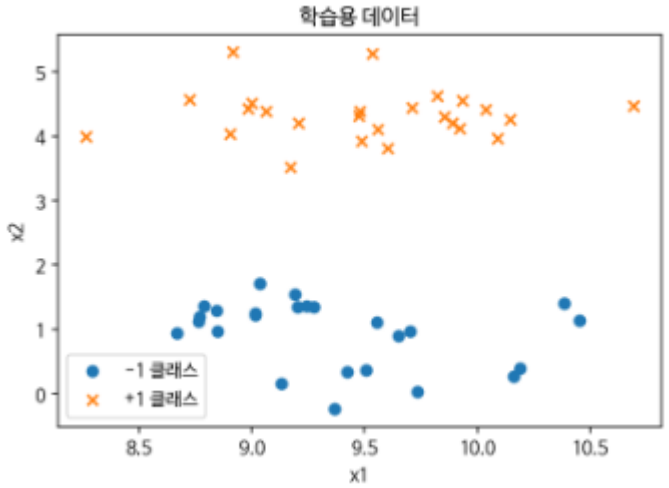
파라미터	default	설명
C	1.0	오류를 얼마나 허용할 것인지 (규제항) 클수록 하드마진, 작을수록 소프트마진에 가까움
kernel	'rbf' (가우시안 커널)	'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'
degree	3	다항식 커널의 차수 결정
gamma	'scale'	결정경계를 얼마나 유연하게 그릴지 결정 클수록 오버피팅 발생 가능성 높아짐
coef0	0.0	다항식 커널에 있는 상수항 r

1. SVM(Support Vector Machine)

Scikit-Learn의 서포트 벡터 머신

- Scikit-Learn의 svm 서브패키지는 서포트 벡터 머신 모형인 SVC (Support Vector Classifier) 클래스를 제공한다.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=2, cluster_std=0.5, random_state=4)
y = 2 * y - 1

plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', label="-1 클래스")
plt.scatter(X[y == +1, 0], X[y == +1, 1], marker='x', label="+1 클래스")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.title("학습용 데이터")
plt.show()
```



1. SVM(Support Vector Machine)

Scikit-Learn의 서포트 벡터 머신

- VC를 불러올 때 kernel='linear'라고 지정
  - 레이블은 빨간 걸 1, 파란걸 0으로 보면 된다.그리고 .fit() 안에 학습 데이터와 레이블을 넣는다

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear')
training_points = [[1, 2], [1, 5], [2, 2], [7, 5], [9, 4], [8, 2]]
labels = [1, 1, 1, 0, 0, 0]
classifier.fit(training_points, labels)

>>

SVC
SVC(kernel='linear')
```

1. SVM(Support Vector Machine)

Scikit-Learn의 서포트 벡터 머신

- predict() 메서드를 통해 분류를 해볼 수 있다. 예를 들어 [3, 2]라는 데이터를 넣어 예측
- [3, 2] 좌표를 찍어봐도 알 수 있듯 빨간 점, 1로 분류

```
print(classifier.predict([[3, 2]]))

>>

[1]
```

- 서포트 벡터, 결정 경계를 정의하는 서포트 벡터를 확인

```
print(classifier.support_vectors_)
>>
[[7. 5.]
 [8. 2.]
 [2. 2.]]
```

1. SVM(Support Vector Machine)

Scikit-Learn의 서포트 벡터 머신

- 서포트 벡터, 결정 경계를 정의하는 서포트 벡터를 평가 점수 확인

```
print("학습 데이터 점수: {}".format(classifier.score(training_points , labels )))
```

```
>>
```

```
학습 데이터 점수: 1.0
```

1. SVM(Support Vector Machine)

붓꽃 문제에의 응용

```
import numpy as np
import matplotlib as mp
import matplotlib.pyplot as plt


from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sk□learn.preprocessing import StandardScaler

iris = load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
>>
```

1. SVM(Support Vector Machine)

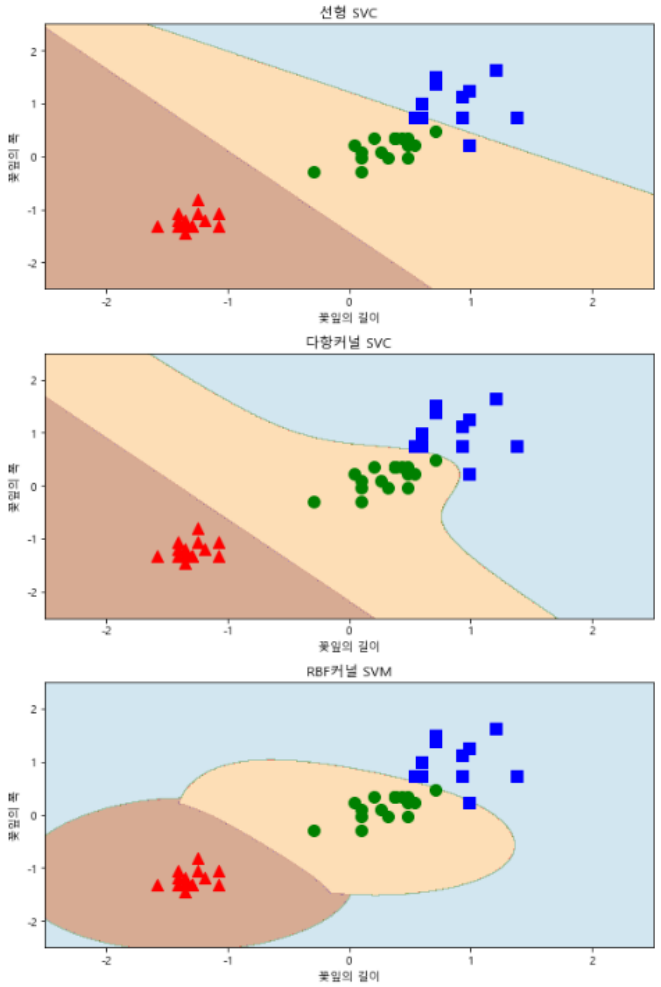
붓꽃 문제에의 응용

```
def plot_iris(X, y, model, title, xmin=-2.5, xmax=2.5, ymin=-2.5, ymax=2.5):
    XX, YY = np.meshgrid(np.arange(xmin, xmax, (xmax-xmin)/1000),
                         np.arange(ymin, ymax, (ymax-ymin)/1000))
    ZZ = np.reshape(model.predict(np.array([XX.ravel(), YY.ravel()]).T), XX.shape)
    plt.contourf(XX, YY, ZZ, cmap=mp.cm.Paired_r, alpha=0.5)
    plt.scatter(X[y == 0, 0], X[y == 0, 1], c='r', marker='^', label='0', s=100)
    plt.scatter(X[y == 1, 0], X[y == 1, 1], c='g', marker='o', label='1', s=100)
    plt.scatter(X[y == 2, 0], X[y == 2, 1], c='b', marker='s', label='2', s=100)
    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)
    plt.xlabel("꽃잎의 길이")
    plt.ylabel("꽃잎의 폭")
    plt.title(title)


model1 = SVC(kernel='linear').fit(X_test_std, y_test)
model2 = SVC(kernel='poly', random_state=0,
             gamma=10, C=1.0).fit(X_test_std, y_test)
```

1. SVM(Support Vector Machine)

붓꽃 문제에의 응용

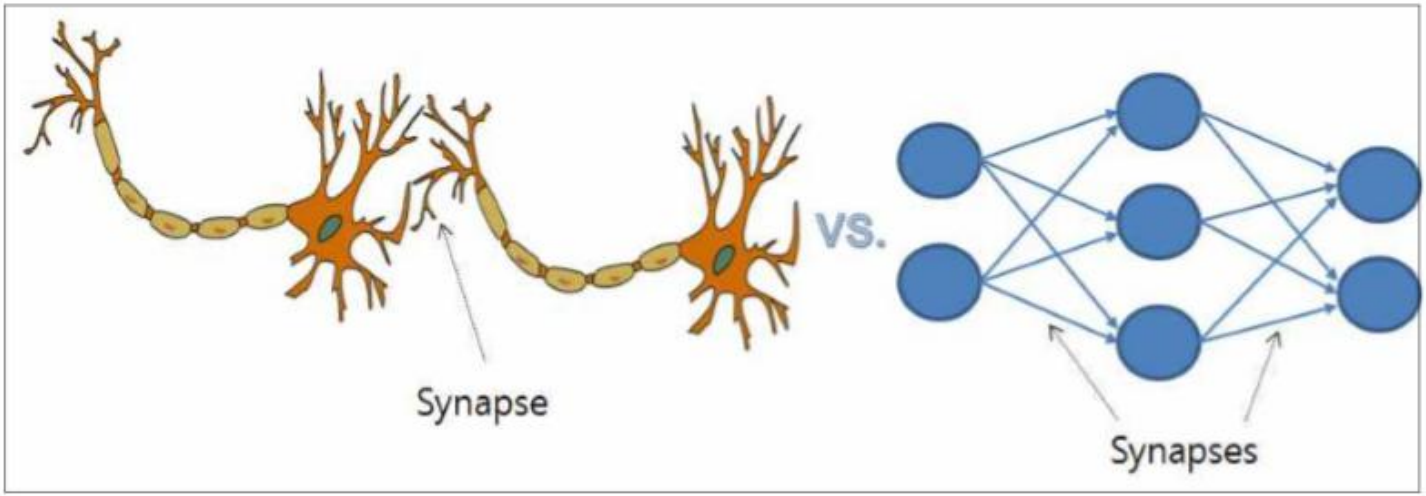
```
model3 = SVC(kernel='rbf', random_state=0, gamma=1,
             C=1.0).fit(X_test_std, y_test)

plt.figure(figsize=(8, 12))
plt.subplot(311)
plot_iris(X_test_std, y_test, model1, "선형 SVC")
plt.subplot(312)
plot_iris(X_test_std, y_test, model2, "다항커널 SVC")
plt.subplot(313)
plot_iris(X_test_std, y_test, model3, "RBF커널 SVM")
plt.tight_layout()
plt.show()
```



2. 인공 신경망 분석

개념

- 인공 신경망 분석 모델은 생물체의 뇌가 감각 입력 자극에 어떻게 반응하는지에 대한 이해로부터 얻은 힌트를 바탕으로 생물체의 신경망을 모사하여, 입력 신호와 출력 신호 간의 관계를 모델화하는 기법이다. 뇌가 뉴런이라는 세포들의 방대한 연결을 통해 신호를 처리하듯, 인공 신경망은 이를 모사한 인공 뉴런(노드)의 네트워크를 구성하여 모델화한다. 아래의 그림 은 생물체의 신경망과 인공 신경망 구조를 비교한 그림이다.

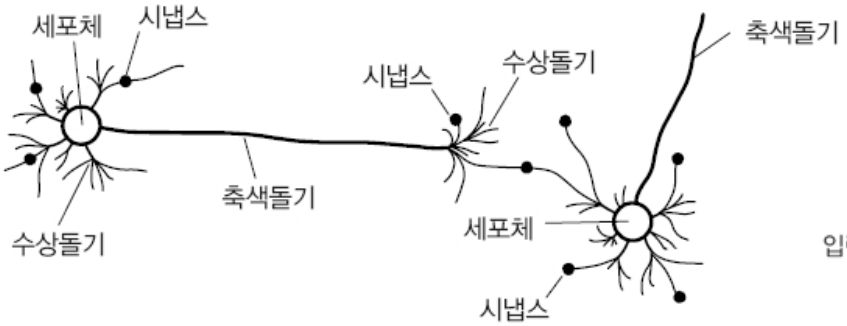


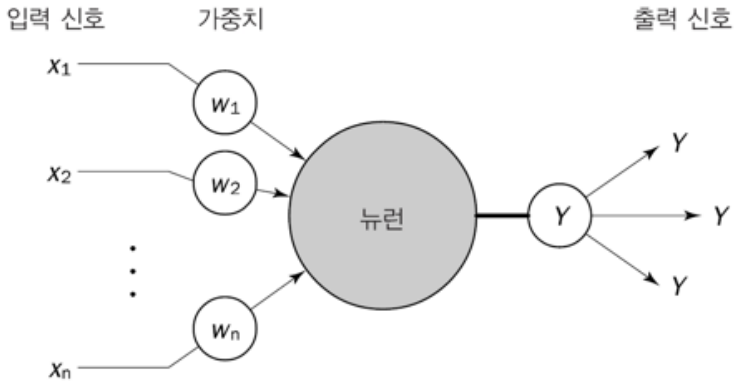
2. 인공 신경망 분석

개념

- 인간 뇌를 기반으로 한 추론 모델.
  - 인간 뇌의 추론 모델 - 뉴런(neuron)
  - 인간의 뇌 모델링

생물학적인 신경망	인공 신경망
세포체	뉴런
수상돌기	입력
축색돌기	출력
시냅스	가중치

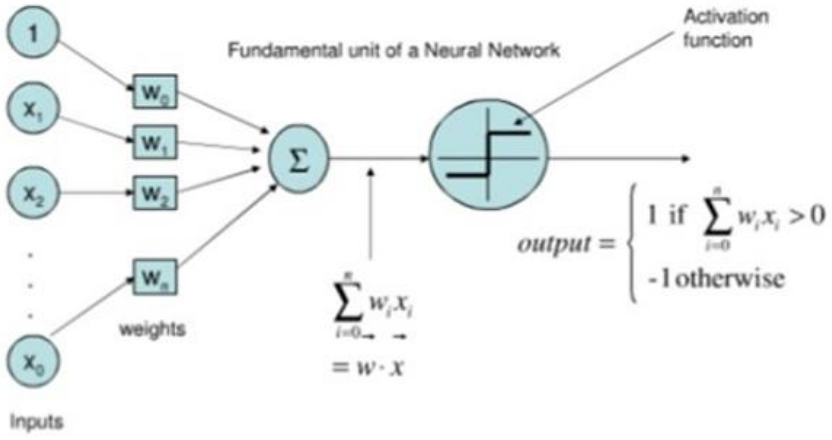




2. 인공 신경망 분석

개념

- 뉴런의 출력 결정
  - 렌 맥클록(Warren McCulloch)과 월터 피츠(Walter Pitts)가 제안 (1943년).
- <A logical calculus of ideas immanent in nervous activity> 각 신경세포 (neuron) 의 기능은 매우 단순하나, 이들이 상호 연결됨으로써 복잡한 계산을 수행하는 신경 시스템의 기초를 마련한 이 논문에서, 현대 컴퓨터의 기반을 이루는 모든 Boolean 논리 표현은 2 진 출력을 갖는 맥클로치 - 피츠 신경세포로 구현 가능함을 보여 주었다.
- 뉴런은 전이 함수, 즉 활성화 함수(activation function)를 사용



$$Y = sign\left[\sum_{i=1}^{n} x_i w_i - \theta\right]$$

$$Y = \begin{cases} +1 & \text{if } X \geq \theta \\ -1 & \text{if } X < \theta \end{cases}$$

2. 인공 신경망 분석

인공 신경망 분석기법의 장/단점

장 점	단 점
✓ 선형적으로 분류할 수 없는 복잡한 비선형 문제에 탁월한 성능을 보인다. ✓ 분류 및 수치예측 문제에 모두 적용 가능하다. ✓ 통계적 기본 가정이 적고, 유연한 모델을 만든다. ✓ 데이터 사이즈가 작거나, 불완전 데이터, 노이즈 데이터가 많은 경우에도 다른 모델에 비하여 예측 성능이 우수한 경우가 많다.	✓ 모델결과 해석이 어려워서 은닉층의 노드들이 무엇 을 표현하는지, 왜 그러한 결과값이 나왔는지 설명이 필요한 모델링에는 적합하지 않다. (블랙박스 모형). ✓ 나이브 베이즈나 로지스틱 회귀 모형 같은 보다 단순한 분류 알고리즘에 비하여 컴퓨팅 연산에 많은 자원이 필요하다. ✓ 과적합화나 과소적합이 발생하기 쉽다. ✓ 전체적 관점에서의 최적해가 아닌 지역 최적해가 선택될 수 있다.

2. 인공 신경망 분석

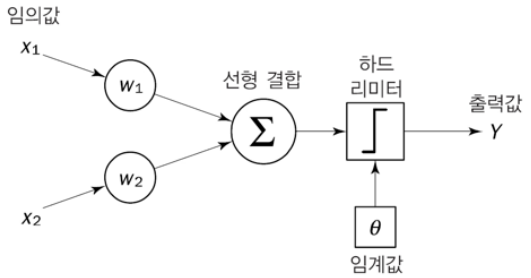
단일 계층 퍼셉트론 : SLP: Single Layer Perceptron (로젠블렛 퍼셉트론)

- 학습능력을 갖는 패턴분류장치. 1957년 프랭크 로젠블랫에 의해 제안
- <The perceptron : a probabilistic model for information storage and organization in the brain> 이라는 논문과 1962년에 발표된《Principles of Neurodynamics》라는 책에서, 신경세포와 유사한 단순 계산기능을 갖는 요소로 구성된 입력층과 출력층을 갖는 perceptron 이라는 신경 시스템의 모델을 제시하고, 입력과 출력 사이의 synapse 를 출력층의 제곱오차가 최소가 되는 방법으로 학습시킬 수 있음을 보여 주었다.

2. 인공 신경망 분석

단일 계층 퍼셉트론 : SLP: Single Layer Perceptron (로젠블렛 퍼셉트론)

- 입력 노드가 두 개인 단층 퍼셉트론



- 로젠블랫퍼셉트론의 동작 원리는 맥클록과피츠의 뉴런 모델에 기반.
- 퍼셉트론은 선형 결합기와 하드 리미터로 구성.
- 입력의 가중합을 하드 리미터에 입력하고, 입력이 양이면 '+1', 음이면 '-1'을 출력.
- 기본적인 퍼셉트론의 경우, 초평면(hyperplane)으로 n차원 공간을 두 개의 결정 영역으로 나뉜다.
- 초평면을 선형 분리 함수로 정의한다.
- 선형 분리 함수 : 식 $\sum_{i=1}^{n} x_i w_i - \theta = 0$

- 임계값θ는 결정 경계를 옮기는 데 쓰인다.
- 분할 식

  입력이 두 개인 퍼셉트론$x_1 w_1 + x_2 w_2 - \theta = 0$

  입력이 세 개인 퍼셉트론$x_1 w_1 + x_2 w_2 + x_3 w_3 - \theta = 0$

2. 인공 신경망 분석

활성화 함수의 등장

- 활성화 함수 h(x): 입력 신호의 총합을 출력 신호로 변환하는 함수. 활성화를 일으키는 지 정하는 역할.

$$y = h(b + w_1x_1 + w_2x_2)$$

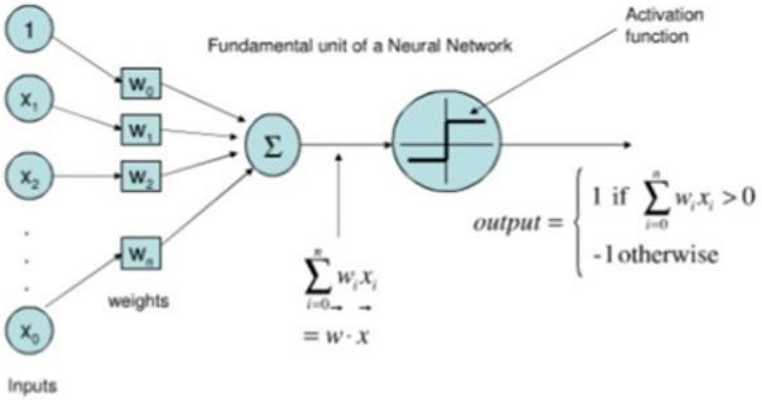
$$h(x) = 0 (x <= 0)$$
$$h(x) = 1 (x > 0)$$

  - 가중치가 달린 입력 신호와 편향의 총합을 계산
    - y=h(a)
  - a를 h(a) 함수에 넣어 y를 출력하는 흐름
  - 단순 퍼셉트론은 단층 네트워크에서 계단 함수를 활성화 함수로 사용한 모델을 가리킴.
  - 다층 퍼셉트론은 신경망(여러 층으로 구성되고 시그모이드 함수 등의 매끈한 활성화 함수를 사용하는 네트워크)을 가리킴
  - 왼쪽은 일반적인 뉴런, 오른쪽은 활성화 처리 과정을 명시한 뉴런

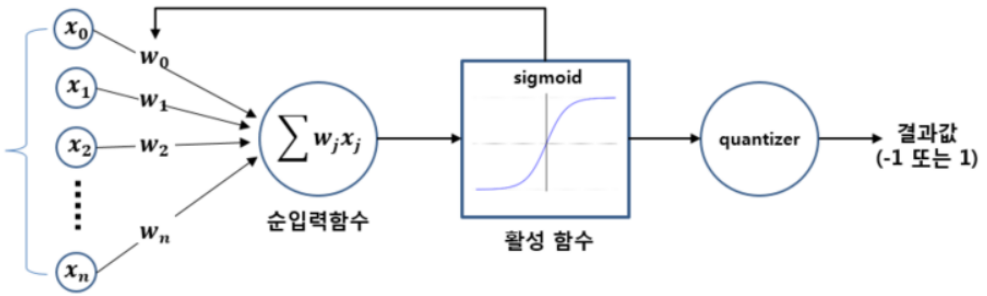
2. 인공 신경망 분석

활성화 함수의 등장

- "퍼셉트론에서는 활성화 함수로 계단 함수를 이용한다" 라고 할 수 있음. 가장 일반적인 활성화 함수로는 계단, 부호, 선형, 시그모이드 함수가 있다.
- 신경망에서는 활성화 함수로 시그모이드 함수를 이용하여 신호를 변환하고, 그 변환된 신호를 다음 뉴런에 전달



$$h(x) = \begin{cases} 0 & (x \le 0) \\ 1 & (x > 0) \end{cases}$$

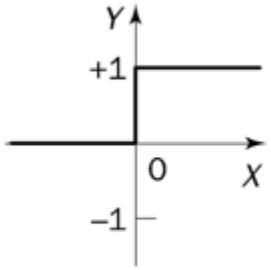


2. 인공 신경망 분석

활성화 함수의 등장

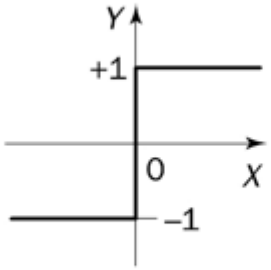
- 가장 일반적인 활성화 함수로는 계단, 부호, 선형, 시그모이드 함수가 있다.

계단 함수



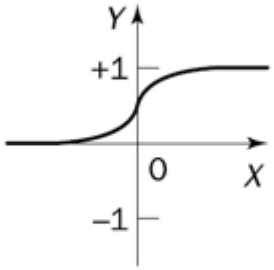
$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

부호 함수



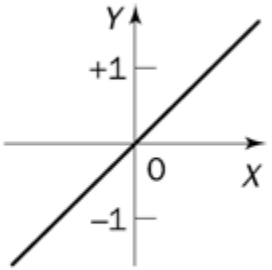
$$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$$

시그모이드 함수



$$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$$

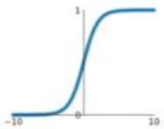
선형 함수



$$Y^{linear} = X$$

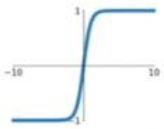
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



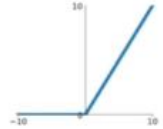
tanh

$\tanh(x)$



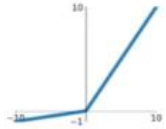
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

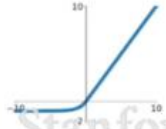


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- 데이터 셋 로드

```
# sklearn의 datasets에서 load_iris를 로드
from sklearn.datasets import load_iris
# iris데이터셋을 iris라는 변수에 저장
iris = load_iris()

# iris에 있는 key값을 나타냄
iris.keys()
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

```
# iris의 데이터에 해당하는 부분의 X와 Y의 크기를 나타냄
iris['data'].shape
```

```
(150, 4)
```

2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- 데이터 셋 로드

```
# iris데이터셋의 0번째부터 9번째까지를 슬라이싱해서 나타냄
iris['data'][0:10]
```

`

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

`

2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- 데이터 셋 분리

```
# X에는 iris데이터의 값 150x4의 크기를 입력
# y에는 분류하고자 하는 target변수를 입력
# target변수는 데이터가 무엇인지에 대해 판별하는 값
# iris target의 경우 0, 1, 2로 구분됨
X = iris['data']
y = iris['target']

# 위의 데이터를 train과 test로 구분
# sklearn의 model_selection 내에 train_test_split를 로드
# train_test_split를 이용해 위의 X변수에 선언한 data값과 y변수에 선언한 target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- 데이터 셋 표준화와 정규화

```
# sklearn 내에 preprocessing의 StandardScaler를 로드
# StandardScaler는 정규화를 시키는 함수
# StandardScaler는 데이터의 범위를 평균 0, 표준편차 1의 범위로 바꿔주는 함수
# 그리고 StandardScaler를 scaler라는 변수에 저장해 사용
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# StandardScaler를 담은 변수에 X_train을 학습해 데이터를 정규화
scaler.fit(X_train)

# X_train과 X_test를 StandardScaler를 이용해 정규화
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- MLP 알고리즘 로드 및 히든 레이어 할당

```
# 다중인공신경망(MLP) 분류 알고리즘을 sklearn의 neural_network에서 로드
from sklearn.neural_network import MLPClassifier

# MLP 알고리즘의 히든레이어를 3계층(10,10,10)으로 할당
mlp = MLPClassifier(hidden_layer_sizes=(10,10,10))
```

2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- 데이터 학습과 학습 성능 평가

```
# 위에서 분류한 X_train과 y_train을 MLP를 이용해 학습
mlp.fit(X_train, y_train)

# mlp로 학습한 내용을 X_test에 대해 예측하여 predictions변수에 저장
predictions = mlp.predict(X_test)

# sklearn.metrics의 confusion_matrix와 classification_report를 로드
# confusion_matrix는 데이터가 맞는지의 유무를 판단
# classification_report는 precision과 recall 그리고 f1_score등을 계산해 정확률에 대해 계산
from sklearn.metrics import classification_report, confusion_matrix

# confusion_matrix를 이용해 실제값 y_test와 예측값에 대해 비교
print(confusion_matrix(y_test, predictions))

`

[[10  0  0]
 [ 0 10  0]
 [ 0  3 15]]`
```

2. 인공 신경망 분석

인공신경망을 이용한 데이터분석

- 정확률, 재현율, f1-score를 확인

```
# classification_report를 이용해 정확률, 재현율, f1-score를 확인
print(classification_report(y_test, predictions))

>>
 precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.77      1.00      0.87        10
           2       1.00      0.83      0.91        18

    accuracy                           0.92        38
   macro avg       0.92      0.94      0.93        38
weighted avg       0.94      0.92      0.92        38
```

THANK YOU.