Euron 1주차 필사

CH1. 파이썬 기반의 머신러닝 생태계 이해

▼ 01. 머신러닝의 개념

- 머신러닝
  - 애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법
  - 업무적으로 복잡한 조건/규칙들이 다양한 형태로 결합하고 시시각각 변하면서 소프트웨어 코드로 로직을 구성하여 이를 관통하는 일정한 패턴을 찾기 어려운 경우 훌륭한 솔루션 제공
  - 데이터를 기반으로 숨겨진 패턴을 인지해 해결
  - 데이터를 관통하는 패턴을 학습, 이에 기반한 예측 수행 → 데이터 분석 영역에 새로운 혁신 가져옴
- 머신러닝 알고리즘
  - 데이터를 기반으로 통계적인 신뢰도 강화
  - 예측 오류를 최소화하기 위한 다양한 수학적 기법을 적용해 데이터 내의 패턴 스스로 인지
  - 신뢰도 있는 예측 결과 도출
- 머신러닝의 분류
  - 지도학습(Supervised Learning)
    - **분류(Classification)**
    - **회귀(Regression)**
    - 추천 시스템
    - 시각/음성 감지/인지텍스트 분석, NLP
  - 비지도학습(Un-supervised Learning)
    - 클러스터링
    - 차원 축소
    - 강화 학습(Reinforcement Learning)

- 머신러닝에서 데이터 vs 머신러닝 알고리즘?
    - 데이터의 중요성이 더 큼
    - 머신러닝 단점 : 데이터에 매우 의존적, 좋은 품질의 데이터를 갖추지 못한다면 머신러닝의 수행 결과도 좋을 수 없음
    - 데이터를 이해하고 효율적으로 가공, 처리, 추출해 최적의 데이터를 기반으로 알고리즘을 구동할 수 있도록 준비하는 능력이 중요
- 파이썬 vs R
    - R
        - 통계 전용 프로그램 언어
        - 개발 언어에 익숙하지 않으나 통계 분석에 능한 사람에게 추천
        - 다양하고 많은 통계 패키지를 보유하고 있음
    - 파이썬
        - 개발 전문 프로그램 언어
        - 머신러닝을 시작하는 사람에게 추천
        - **파이썬은 소리 없이 프로그래밍 세계를 점령하고 있는 언어**
        - 쉽고 뛰어난 개발 생산성
        - 많은 라이브러리로 인해 개발 시 높은 생산성 보장
        - 뛰어난 확장성, 유연성, 호환성으로 인해 서버, 네트워크, 시스템, IOT, 데스크톱 등 다양한 영역에서 사용됨
        - 머신러닝 애플리케이션과 결합한 다양한 애플리케이션 개발 가능
        - **딥러닝 프레임워크**인 TensorFlow, Keras, PyTorch 등에서 파이썬 우선 정책으로 파이썬 지원

▼ 02. 파이썬 머신러닝 생태계를 구성하는 주요 패키지

- 머신러닝 패키지 : 대표적인 머신러닝 패키지는 사이킷런(Scikit-Learn)
- 행렬/선형대수/통계 패키지 : 넘파이(Numpy)
- 데이터 핸들링 : 판다스(2차원 데이터 처리에 특화, 넘파이보다 훨씬 편리하게 데이터 처리를 할 수 있는 많은 기능 제공)
- 시각화 : 맷플롯립(Matplotlib), 시본(Seaborn)

▼ 03. 넘파이(Numpy)

- Numerical Python을 의미
- 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지
- 장점
  - 루프를 사용하지 않고 대량 데이터의 배열 연산을 가능하게 함 → 빠른 배열 연산 속도 보장
  - 많은 과학과 공학 패키지가 넘파이에 의존하고 있음
  - C/C++과 같은 저수준 언어 기반의 호환 API를 제공
- 일반적으로 **데이터는 2차원 형태의 행과 열**로 이루어졌으며, 이에 대한 다양한 가공과 변환, 여러 가지 통계용 함수의 적용 등이 필요 → 편리성 한계 존재
- 넘파이 ndarray 개요
  - 넘파이 기반 데이터 타입 : ndarray
  - 다차원 배열을 쉽게 생성하고 다양한 연산을 수행할 수 있음
  - array() 함수
    - 파이썬의 리스트와 같은 다양한 인자를 입력받아서 ndarray로 변환하는 기능 수행
  - shape 변수
    - ndarray의 크기, 즉 행과 열의 수를 튜플 형태로 가지고 있음
    - ndarray 배열의 차원까지 알 수 있음
  - np.array()
    - ndarray로 변환을 원하는 객체를 인자로 입력하면 ndarray를 반환
  - ndarray.shape
    - ndarray의 차원과 크기를 튜플 형태로 나타내 줌
    - [1,2,3]인 array1의 shape은 (3,) : 1차원 array로 3개의 데이터를 가지고 있음
    - [[1,2,3], [2,3,4]]인 array2의 shape는 (2,3) : 2차원 array로 2개의 row와 3개의 column으로 구성됨 → 2*3=6개의 데이터를 가지고 있음을 의미
    - [[1,2,3]]인 array3 : shape는 (1,3) → 1개의 로우와 3개의 칼럼으로 구성된 2차원 데이터

        - 명확하게 차원의 차수를 변환하는 방법을 알아야 차원이 달라서 발생하는 오류 방지 가능
    - ndarray.ndim
        - 각 array 차원을 확인할 수 있음
        - array() 함수의 인자로는 파이썬 리스트 객체가 주로 사용됨
            - 리스트 []는 1차원, 리스트의 리스트 [[]]는 2차원과 같은 형태로 배열의 차원과 크기를 쉽게 표현 가능
- ndarray의 데이터 타입
    - 숫자, 문자열, 불 값 등이 모두 가능
    - 숫자형은 int, unsigned int, float형, complex 타입도 제공
    - 연산의 특성상 같은 데이터 타입만 가능
        - 한 개의 ndarray 객체에 int와 float이 함께 있을 수 없음
        - 만약 다른 데이터 유형이 섞여 있는 리스트를 ndarray로 변경하면 데이터 크기가 더 큰 데이터 타입으로 형 변환 일괄 적용
            - ex) int형과 string형이 섞여 있는 리스트 → string으로 바뀜
            - ex2) int형과 float형이 섞여 있는 리스트 → float형으로 바뀜
    - dtype 속성으로 확인 가능
    - astype()
        - 데이터값의 타입 변경
        - 인자로 원하는 타입을 문자열로 지정하면 됨
        - 메모리를 더 절약해야 할 때 보통 이용됨
            - 예) int형으로 충분한 경우인데, 데이터 타입이 float라면 int형으로 바꿔서 메모리를 절약할 수 있음
- ndarray 편리하게 생성하기
    - 주로 테스트용으로 데이터를 만들거나 대규모의 데이터를 일괄적으로 초기화해야 할 경우에 사용됨
    - arange()
        - 파이썬 표준 함수인 range()와 유사한 기능을 함
        - array를 range()로 표현하는 것

        - 0부터 함수 인자 값 -1까지의 값을 순차적으로 ndarray의 데이터값으로 변환해 줌
        - default 함수 인자는 stop 값, start 값도 부여해 0이 아닌 다른 값부터 시작한 연속 값을 부여할 수도 있음
    - zeros()
        - 함수 인자로 튜플 형태의 shape 값을 입력하면 **모든 값을 0**으로 채운 해당 shape를 가진 ndarray를 반환
        - dtype으로 함수 인자를 정해주지 않으면 default로 float64 형의 데이터로 ndarray를 채움
    - ones()
        - 함수 인자로 튜플 형태의 shape 값을 입력하면 **모든 값을 1**으로 채운 해당 shape를 가진 ndarray를 반환
        - dtype으로 함수 인자를 정해주지 않으면 default로 float64 형의 데이터로 ndarray를 채움
- ndarray의 차원과 크기를 변경하는 reshape()
    - ndarray를 특정 차원 및 크기로 변환
    - 변환을 원하는 크기를 함수 인자로 부여하면 됨
    - 지정된 사이즈로 변경이 불가능하면 오류 발생
    - 인자 -1을 적용해 실전에서 더 효율적으로 사용 가능
        - 원래 ndarray와 호환되는 새로운 shape로 변환해 줌
        - 예) 1차원 ndarray로 0~9까지의 데이터를 가진 array1을 array1.reshape(-1, 5)를 써서 2차원 ndarray로 변환 가능 → 고정된 5개의 카럼에 맞는 로우를 자동으로 새롭게 생성해 변환하라는 의미
        - -1을 사용하더라도 호환될 수 없는 형태는 변환할 수 없음
        - reshape(-1,1)와 같은 형태로 자주 사용됨
            - 원본 ndarray가 어떤 형태라도 2차원이고, 여러 개의 로우를 가지되 반드시 1개의 칼럼을 가진 ndarray로 변환됨을 보장
- 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(Indexing)
    - 특정한 데이터만 추출 : 원하는 위치의 인덱스 값을 지정하면 해당 위치의 데이터가 반환됨

- 단일 값 추출 : 한 개의 데이터만 추출
  - ndarray 객체에 해당하는 위치의 인덱스 값을 [] 안에 입력
  - 예) array1[2]의 타입은 ndarray 타입이 아니라 ndarray 내의 데이터값을 의미
  - 인덱스에 마이너스 기호를 이용하면 맨 뒤에서부터 데이터를 추출할 수 있음
  - 단일 인덱스를 이용해 데이터값도 간단히 수정 가능
  - 다차원 ndarray에서 단일 값 추출
    - 2차원의 경우 콤마로 분리된 로우와 칼럼 위치의 인덱스를 통해 접근
    - 예) array2d[1,0]은 두 번째 로우, 첫 번째 칼럼 위치의 데이터를 가리킴
  - axis 0 : 로우 방향의 축, axis 1 : 칼럼 방향의 축
    - ndarray의 정확한 표현은 로우, 칼럼이 아닌 axis 0, axis 1이 맞음
    - 즉, 인덱싱은 [axis 0=0, axis 1=1]이 정확한 표현
    - 2차원은 axis 0, axis 1 / 3차원은 axis 0, axis 1, axis 2(행, 열, 높이)
- 슬라이싱 : : 기호 사이에 시작 인덱스와 종료 인덱스를 표시하면 시작 인덱스에서 종료 인덱스-1 위치에 있는 데이터 ndarray를 반환
  - 슬라이싱으로 추출된 데이터 세트는 모두 ndarray 타입
  - : 사이의 시작, 종료 인덱스는 생략 가능
    - 2차원은 1차원과 유사하지만 단지 콤마로 로우와 칼럼 인덱스를 지칭하는 부분만 다름
    - 로우나 칼럼 축 한쪽에만 슬라이싱을 적용하고 다른 쪽 축에는 단일 값 인덱스를 적용해도 됨
    - 2차원 ndarray에서 뒤에 오는 인덱스를 없애면 1차원 ndarray를 반환함
    - 3차원 ndarray에서 뒤에 오는 인덱스를 없애면 2차원 ndarray를 반환함
- 팬시 인덱싱 : 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray를 반환
  - 예1) array2d[[0,1], 2] → (row, col) (0,2), (1,2)로 적용됨

        - 예2) array2d[[0,1], 0:2] → (row, col) ((0,0),(0,1)), ((1,0),(1,1))로 적용됨
        - 예3) array2d[[0,1]] → (row, col) ((0,:), (1,:)) 로 적용됨
    - 불린 인덱싱 : 특정 조건에 해당하는지 여부인 True/False 값 인덱싱 집합을 기반으로 True에 해당하는 인덱스 위치에 있는 데이터의 ndarray를 반환
        - 조건 필터링과 검색을 동시에 할 수 있기 때문에 매우 자주 사용됨
        - ndarray의 인덱스를 지정하는 [ ] 내에 조건문을 그대로 기재하면 됨
            - 예) array1d>5 같이 ndarray 객체에 조건식만 붙이면 False, True로 이뤄진 ndarray 객체가 반환됨
            - 조건으로 반환된 이 ndarray 객체를 인덱싱을 지정하는 [ ] 내에 입력하면 False 값은 무시하고 True 값이 있는 위치 인덱스 값으로 자동 변환해 해당하는 인덱스 위치의 데이터만 반환하게 됨
        - 동작하는 단계
            - 1단계 : array1d>5와 같이 ndarray의 필터링 조건을 [ ] 안에 기재
            - 2단계 : False 값은 무시하고 True 값에 해당하는 인덱스값만 저장(유의해야 할 사항은 True값 자체인 1을 저장하는 것이 아니라 True값을 가진 인덱스를 저장한다는 것)
            - 3단계 : 저장된 인덱스 데이터 세트로 ndarray 조회
- 행렬의 정렬 sort()와 argsort()
    - np.sort() vs ndarray.sort()
        - 넘파이에서 sort()를 호출하는 방식 vs 행렬 자체에서 sort()를 호출하는 방식
        - 원 행렬은 그대로 유지한 채 원 행렬의 정렬된 행렬 반환 vs 행렬 자체를 정렬한 형태로 변환, 반환 값은 None
        - 모두 기본적으로 오름차순으로 행렬 내 원소를 정렬함
        - 내림차순으로 정렬하기 위해서는 [::-1]을 적용
        - 행렬이 2차원 이상일 경우 axis 축 값 설정을 통해 로우 방향, 또는 칼럼 방향으로 정렬 수행 가능
    - 정렬된 행렬의 인덱스 반환
        - np.argsort()

- 원본 행렬이 정렬되었을 때 기존 원본 행렬의 원소에 대한 인덱스를 필요로 할 때 사용
- 정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 반환
- 내림차순으로 정렬 원하면 np.argsort()[::-1]과 같이 쓰면 됨
- 넘파이에서 매우 활용도가 높음(ndarray는 메타 데이터를 가질 수 없음 → 실제 값과 그 값을 뜻하는 메타 데이터를 별도의 ndarray로 각각 가져야만 함)
  - 예) 학생별 시험 성적을 데이터로 표현하기 위해서는 학생의 이름과 시험 성적을 각각 ndarray로 가져야 함 → np.argsort(score_array)를 이용해 시험 성적순으로 학생 이름 출력 가능

- 선형대수 연산 - 행렬 내적과 전치 행렬 구하기
  - 행렬 내적(행렬 곱)
    - np.dot()
      - 두 행렬 A와 B의 내적 계산 가능
      - 왼쪽 행렬의 행과 오른쪽 행렬의 열의 원소들을 순차적으로 곱한 뒤 그 결과를 모두 더한 값
      - 왼쪽 행렬의 열 개수와 오른쪽 행렬의 행 개수가 동일해야 내적 연산 가능
  - 전치 행렬
    - np.transpose()
      - 원 행렬에서 행과 열 위치를 교환한 원소로 구성한 행렬
      - A^T와 같이 표기함

▼ 04. 데이터 핸들링 - 판다스

- 판다스의 유래
  - 금융회사의 분석 전문가인 웨스 매키니가 회사에서 사용하는 분석용 데이터 핸들링 툴이 마음에 들지 않아 판다스를 개발함
- 파이썬에서 데이터 처리를 위해 존재하는 가장 인기 있는 라이브러리
- 2차원 데이터를 효율적으로 가공/처리할 수 있는 다양하고 훌륭한 기능 제공

- 고수준 API를 제공, 파이썬의 리스트와 컬렉션과 넘파이 등의 내부 데이터뿐만 아니라 CSV 등의 파일을 쉽게 DataFrame으로 변경해 데이터의 가공/분석을 편리하게 수행할 수 있게 만듦
- 판다스의 핵심 객체 : DataFrame
  - 여러 개의 행과 열로 이뤄진 2차원 데이터를 담는 데이터 구조체
- Index
  - 개별 데이터를 고유하게 식별하는 Key 값
- Series vs DataFrame
  - 둘 다 Index를 key 값으로 가짐
  - 칼럼이 하나뿐인 데이터 구조체 vs 칼럼이 여러 개인 데이터 구조체
- cdv, tab과 같은 다양한 유형의 분리 문자로 칼럼을 분리한 파일을 손쉽게 DataFrame으로 로딩할 수 있게 해줌
- 판다스 시작 - 파일을 DataFrame으로 로딩, 기본 API
  - pandas를 pd로 alias해 임포트하는 것이 관례
  - read_csv()
    - CSV(칼럼을 ‘,’로 구분한 파일 포맷) 파일 포맷 변환을 위한 API
    - CSV뿐만 아니라 어떤 필드 구분 문자 기반의 파일 포맷도 DataFrame으로 변환 가능
    - 필드 구분 문자가 콤마(,)
    - 인자인 sep에 해당 구분 문자 입력하면 됨
      - 예) read_csv(’파일명’, sep=’\t’)
    - filepath 인자가 가장 중요, 나머지 인자는 지정하지 않으면 디폴트 값으로 할당됨
      - 로드하려는 데이터 파일의 경로를 포함한 파일명을 입력하면 됨
      - 파일명만 입력하면 파이썬 실행 파일이 있는 디렉터리와 동일한 디렉터리에 있는 파일명을 로딩
    - 별다른 파라미터 지정이 없으면 파일의 맨 처음 로우를 칼럼명으로 인지하고 칼럼으로 변환
  - read_table()

        - 필드 구분 문자가 탭('\t')
    - read_fwf()
        - 고정 길이 기반의 칼럼 포맷을 DataFrame으로 로딩하기 위한 API
    - 모든 DataFrame 내의 데이터는 생성되는 순간 고유의 Index 값을 가짐
    - DataFrame.head()
        - DataFrame의 맨 앞에 있는 N개의 로우를 반환(Default는 5개)
    - DataFrame.shape
        - 데이터프레임의 행과 열 크기를 알아볼 수 있음
        - 행과 열을 튜플 형태로 반환
    - DataFrame
        - 데이터뿐만 아니라 칼럼의 타입, Null 데이터 개수, 데이터 분포도 등의 메타 데이터 등도 조회 가능
    - DataFrame.info()
        - 총 데이터 건수와 데이터 타입, Null 건수를 알 수 있음
    - DataFrame.describe()
        - 칼럼별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값을 나타냄
        - 오직 숫자형 칼럼의 분포도만 조사, 자동으로 object 타입의 칼럼은 출력에서 제외시킴
        - 개략적인 수준의 분포도를 확인할 수 있어 유용
        - count
            - Not Null인 데이터 건수
        - mean(평균값), std(표준편차), min(최솟값), max(최댓값)
        - 25%,50%,75%는 각각 25,50,75 percentile 값을 의미
        - 카테고리 칼럼을 숫자로 표시할 수도 있음
            - 예) 남을 1, 여를 2
    - DataFrame['칼럼명']
        - Series 형태로 특정 칼럼 데이터 세트가 반환됨

        - value_counts() 메서드를 호출하면 해당 칼럼값의 유형과 건수 확인 가능
            - 많은 건수 순서로 정렬되어 값 반환
    - Series.head()
        - 데이터프레임의 인덱스와 동일한 인덱스가 생성됨
        - Series 객체에서 value_counts() 메서드를 호출하는 것이 칼럼별 데이터 값의 분포도를 더 명시적으로 파악하기 쉬움
            - value_counts()가 반환하는 데이터 타입 역시 Series 객체
            - 맨 왼쪽이 인덱스값, 오른쪽이 데이터값
            - 고유 칼럼 값을 식별자로 사용할 수 있음
    - 인덱스
        - DataFrame, Series가 만들어진 후에도 변경 가능
        - 숫자형뿐만 아니라 문자열도 가능
        - 단, 모든 인덱스는 고유성이 보장되어야 함
    - value_counts()
        - Null 값을 포함하여 개별 데이터 값의 건수를 계산할지를 dropna 인자로 판단
            - dropna의 기본값 : True, Null 값을 무시함
            - Null 값 포함 원하면 dropna의 인자값을 False로 입력하면 됨
- DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환
    - DataFrame ↔ 파이썬 리스트, 딕셔너리, 넘파이 ndarray 등
    - 넘파이 ndarray, 리스트, 딕셔너리 → DataFrame
        - DataFrame은 리스트와 넘파이 ndarray와 달리 칼럼명을 가짐
            - 칼럼명으로 편하게 데이터 핸들링 가능
        - 변환 시 칼럼명을 지정(지정하지 않으면 자동으로 칼럼명 할당)
        - DataFrame 객체의 생성 인자 data는 리스트나 딕셔너리, 넘파이 ndarray를 입력 받음
        - 생성 인자 columns는 칼럼명 리스트를 입력받아 데이터프레임 생성
        - 2차원 이하의 데이터들만 DataFrame으로 변환됨

        - 1차원 형태의 데이터를 기반으로 생성할 경우 칼럼명이 한 개만 필요
        - 2행 3열 형태의 리스트와 ndarray 기반으로 DataFrame 생성? → 칼럼명 3개 필요
      - 딕셔너리 → DataFrame
        - 딕셔너리의 키는 칼럼명으로, 딕셔너리의 값은 키에 해당하는 칼럼 데이터로 변환
        - 키의 경우 문자열, 값의 경우 리스트 또는 ndarray 형태로 딕셔너리를 구성
    - DataFrame → 넘파이 ndarray, 리스트, 딕셔너리
      - 많은 머신러닝 패키지가 기본 데이터 형으로 넘파이를 사용
      - 머신러닝 패키지의 입력 인자 등을 적용하기 위해 다시 넘파이 ndarray로 변환하는 경우가 빈번
      - DataFrame → ndarray : DataFrame 객체의 values를 이용해 쉽게 할 수 있음
      - DataFrame → 리스트, 딕셔너리
        - 리스트로의 변환 : values로 얻은 ndarray에 tolist() 호출
        - 딕셔너리로의 변환 : DataFrame 객체의 to_dict() 메서드 호출
          - 인자로 ‘list’를 입력하면 딕셔너리의 값이 리스트형으로 반환됨
- DataFrame의 칼럼 데이터 세트 생성과 수정
  - DataFrame [ ] 내에 새로운 칼럼명을 입력하고 값을 할당하면 됨
  - Series에 상숫값을 할당하면 Series의 모든 데이터 세트에 일괄 적용됨
  - 기존 칼럼 Series의 데이터를 이용해 새로운 칼럼 Series를 만들 수 있음
  - 업데이트를 원하는 칼럼 Series를 DataFrame[] 내에 칼럼명으로 입력한 뒤에 값을 할당하면 기존 칼럼 값도 쉽게 일괄 업데이트할 수 있음
- DataFrame 데이터 삭제
  - drop() 메서드를 이용
  - drop() 메서드의 원형
    - DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors=’raise’)

        - axis 값에 따라 특정 갈럼 또는 특정 행을 드롭 (axis=0은 로우 방향, 1은 칼럼 방향 축)
        - 대부분의 경우 칼럼을 드롭하기 때문에 axis=1을 씀
        - 이상치 데이터를 삭제하는 경우 axis=0을 씀
    - inplace=False
        - 자신의 데이터프레임 데이터는 삭제하지 않으며, 삭제된 결과 데이터프레임을 반환
    - inplace=True
        - 자신의 DataFrame의 데이터 삭제
        - 반환 값이 None이 됨 → 반환 값을 다시 자신의 DataFrame 객체로 할당하면 안 됨
    - labels
        - 여러 개의 칼럼을 삭제하고 싶으면 리스트 형태로 삭제하고자 하는 칼럼명을 입력
- Index 객체
    - DataFrame, Series의 레코드를 고유하게 식별하는 객체
    - .index 속성으로 Index 객체만 추출 가능
    - 식별성 데이터를 1차원 array로 가지고 있음
    - ndarray와 유사하게 단일 값 반환 및 슬라이싱도 가능
    - 한 번 만들어진 DataFrame 및 Series의 Index 객체는 함부로 변경할 수 없음
    - Series 객체에 연산 함수를 적용할 때 Index를 연산에서 제외됨
    - reset_index()
        - 새롭게 인덱스를 연속 숫자형으로 할당, 기존 인덱스는 ‘index’라는 새로운 칼럼명으로 추가
        - 인덱스가 연속된 int 숫자형 데이터가 아닐 경우 다시 이를 연속 int 숫자형 데이터로 만들 때 주로 사용
        - drop=True로 설정하면 기존 인덱스는 삭제됨
- 데이터 셀렉션 및 필터링
    - DataFrame 뒤에 있는 [ ]는 칼럼만 지정할 수 있는 ‘칼럼 지정 연산자’

    - 넘파이의 [ ] 나 Series의 [ ] 와 다름
    - 인덱스 형태로 반환 가능한 표현식은 [ ] 내에 입력할 수 있음
        - 슬라이싱 연산으로 데이터 추출하는 방법은 사용하지 않는 게 좋음
    - 불린 인덱싱 표현 : [ ] 내 원하는 데이터를 편리하게 추출
    - DataFrame iloc[ ] 연산자
        - 로우나 칼럼을 지정하여 데이터를 선택할 수 있는 인덱싱 방식
            - iloc[ ] : 위치 기반 인덱싱 방식으로 동작
                - 행과 열 위치를, 0을 출발점으로 하는 세로축, 가로축 좌표 정숫값으로 지정
                - 행과 열의 좌표 위치에 해당하는 값으로 정숫값 또는 정수형의 슬라이싱, 팬시 리스트 값을 입력해야 함
                    - 예: iloc[행 위치 정숫값, 열 위치 정숫값]
                - 인덱스 값이나 칼럼명을 입력하면 오류 발생
                - 열 위치에 -1을 입력하여 가장 마지막 열 데이터를 가져오는 데 사용
                - 불린 인덱싱 제공하지 않음
            - loc[ ] : 명칭 기반 인덱싱 방식
                - 데이터 프레임의 인덱스 값으로 행 위치를, 칼럼 명칭으로 열 위치 지정
                - loc[인덱스값, 칼럼명]
                - loc[ ] 에 슬라이싱 기호를 적용하면 종료값까지 포함(종료값-1 아님 주의!!)
                - loc는 iloc과 달리 불린 인덱싱 가능
        - 개별 또는 여러 칼럼 값 전체를 추출하고자 한다면 iloc[]나 loc[]를 사용하지 않고 DataFrame['칼럼명']만으로 충분
        - 행과 열을 함께 사용하여 데이터 추출해야 한다면 iloc[]나 loc[] 사용
- 불린 인덱싱
    - [], loc[]에서 공통으로 지원

    - [ ] 내에 불린 인덱싱을 적용하면 반환되는 객체가 DataFrame이므로 원하는 칼럼명만 별도로 추출 가능
    - 칼럼이 두 개 이상이면 [[ ]] 사용
    - and 조건일 때는 &, or 조건일 때는 |, Not 조건일 때는 ~ 사용
- 정렬, Aggregation 함수, GroupBy 적용
    - DataFrame, Series의 정렬 - sort_values()
        - order by와 매우 유사
        - 주요 입력 파라미터 : by, ascending, inplace
            - by : 특정 칼럼을 입력하면 해당 칼럼으로 정렬 수행
            - ascending=True : 오름차순(기본), False면 내림차순
            - inplace=False(원래 DataFrame은 그대로 유지, 기본), True면 원본 정렬 DataFrame으로 바뀜
    - Aggregation 함수 적용
        - DataFrame에서 바로 aggregation을 호출할 경우 모든 칼럼에 해당 aggregation을 적용
        - count()를 적용하면 Null 값을 반영하지 않은 결과를 반환함
    - groupby() 적용
        - 입력 파라미터 by에 칼럼을 입력하면 대상 칼럼으로 groupby됨
        - groupby()를 호출하면 DataFrameGroupBy라는 또 다른 형태의 DataFrame을 반환
        - groupby()를 호출해 반환된 결과에 aggregation 함수를 호출하면 groupie() 대상 칼럼을 제외한 모든 칼럼에 해당 aggregation 함수 적용
        - 서로 다른 aggregation 함수를 적용할 경우 적용하려는 여러 개의 aggregation 함수명을 DataFrameGroupBy 객체의 agg() 내에 인자로 입력해 사용함
- 결손 데이터 처리하기
    - 결손 데이터 : 칼럼에 값이 없는 Null인 경우, 넘파이의 NaN로 표시
    - 기본적으로 머신러닝 알고리즘은 NaN 값을 처리하지 않으므로 이 값을 다른 값으로 대체해야 함
    - NaN 값은 평균, 총합 등의 함수 연산 시 제외됨

    - isna()
        - NaN 여부를 확인하는 API
        - True, False로 알려줌
        - 결손 데이터 개수는 isna() 결과에 sum() 함수를 추가해 구할 수 있음
            - 내부적으로 True는 1, False는 0으로 변환됨
    - fillna()
        - NaN 값을 다른 값으로 대체하는 API
        - 반환 값을 다시 받거나 inplace=True 파라미터를 fillna()에 추가해야 실제 데이터 세트 값이 변경됨
- apply lambda 식으로 데이터 가공
    - apply 함수에 lambda 식을 결합해 DataFrame이나 Series의 레코드별로 데이터를 가공하는 기능 제공
    - 일괄적으로 데이터 가공하는 것이 속도 면에서 더 빠르나 복잡한 데이터 가공이 필요한 경우 이용
    - : 로 입력 인자와 반환될 입력 인자의 계산식을 분리
        - :의 왼쪽은 입력 인자, 오른쪽은 입력 인자의 계산식
        - 오른쪽의 계산식은 결국 반환 값을 의미
        - 여러 개의 값을 인자로 사용할 경우 map() 결합해 사용
    - lambda 식은 if else를 지원
        - if 식보다 반환 값을 먼저 기술해야 함
        - else의 경우는 else 식이 먼저 나오고 반환 값이 나중에 옴
    - if, else만 지원하고 if, else if, else는 지원하지 않음
    - else if를 이용하기 위해서는 else 절을 ()로 내포해 () 내에서 다시 if else 적용해 사용
    - else if가 많이 나와야 하는 경우나 switch case문의 경우는 아예 별도의 함수를 만드는 게 더 나을 수 있음
-

CH2. 사이킷런으로 시작하는 머신러닝

▼ 01. 사이킷런 소개와 특징

1. 사이킷런 소개와 특징
    a. 파이썬 머신러닝 라이브러리 중 가장 많이 사용되는 라이브러리

▼ 02. 첫 번째 머신러닝 만들어 보기 - 붓꽃 품종 예측하기

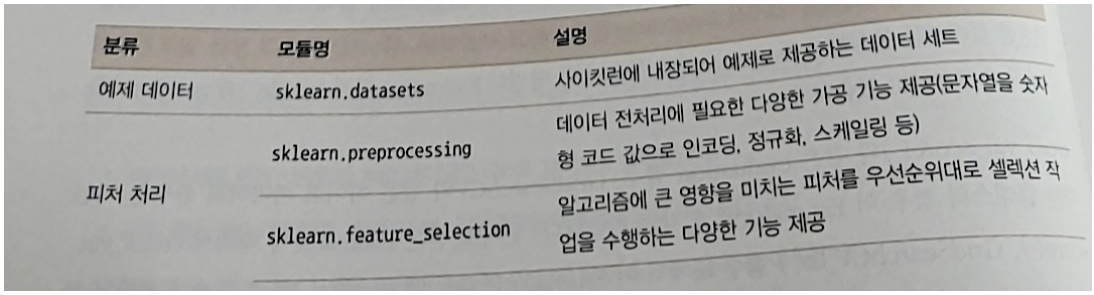
- 붓꽃 데이터 세트로 붓꽃의 품종을 분류해보기
    - 데이터 세트 생성 : load_iris()
    - ML 알고리즘 : 의사 결정 트리 알고리즘으로 DecisionTreeClassifier 적용
        - 객체로 생성하고 생성된 객체의 fit() 메서드에 학습용 피처 데이터 속성과 결정값 데이터 세트를 입력해 호출하면 학습 수행
    - 데이터 세트를 학습 데이터와 테스트 데이터로 분리 : train_test_split()
        - 학습용 데이터와 테스트용 데이터는 반드시 분리해야 함
        - test_size=0.2로 입력 파라미터를 설정하면 전체 데이터 중 테스트 데이터가 20%, 학습 데이터가 80%로 데이터를 분할함
    - 피처들과 데이터 값이 어떻게 구성돼 있는지 확인하기 위해 DataFrame으로 변환
    - DecisionTreeClassifier 객체는 학습 데이터를 기반으로 학습 완료
    - 예측
        - 예측은 반드시 학습 데이터가 아닌 다른 데이터를 이용해야 함, 일반적으로 테스트 데이터 세트 이용
        - predict() 메서드에 테스트용 피처 데이터 세트를 입력해 호출 : 학습된 모델 기반에서 테스트 데이터 세트에 대한 예측값 반환
    - 예측 성능 평가
        - 정확도 : 예측 결과가 실제 레이블 값과 얼마나 정확하게 맞는지 평가하는 지표
            - accuracy_score() 함수 제공
            - 첫 번째 파라미터로 실제 레이블 데이터 세트, 두번째 파라미터로 예측 레이블 데이터 세트를 입력하면 됨
- 지도학습

    - 다양한 피처와 분류 결정값인 레이블 데이터로 모델을 학습한 뒤, 별도의 테스트 데이터 세트에서 미지의 레이블을 예측
    - 명확한 정답이 주어진 데이터를 먼저 학습한 뒤 미지의 정답을 예측
    - 학습을 위해 주어진 데이터 세트 : 학습 데이터 세트
    - 머신러닝 모델의 예측 성능을 평가하기 위해 별도로 주어진 데이터 세트 : 테스트 데이터 세트
- 모듈명은 sklearn으로 시작하는 명명 규칙이 있음
- sklearn.dataset 내의 모듈 : 사이킷런에서 자체적으로 제공하는 데이터 세트를 생성하는 모듈의 모임
- sklearn.tree 내의 모듈 : 트리 기반 ML 알고리즘을 구현한 클래스의 모임
- sklearn.model_selection : 학습 데이터, 검증 데이터, 예측 데이터로 데이터를 분리하거나 최적의 하이퍼 파라미터로 평가하기 위한 다양한 모듈의 모임
    - 하이퍼 파라미터 : 머신러닝 알고리즘별로 최적의 학습을 위해 직접 입력하는 파라미터 통칭
-

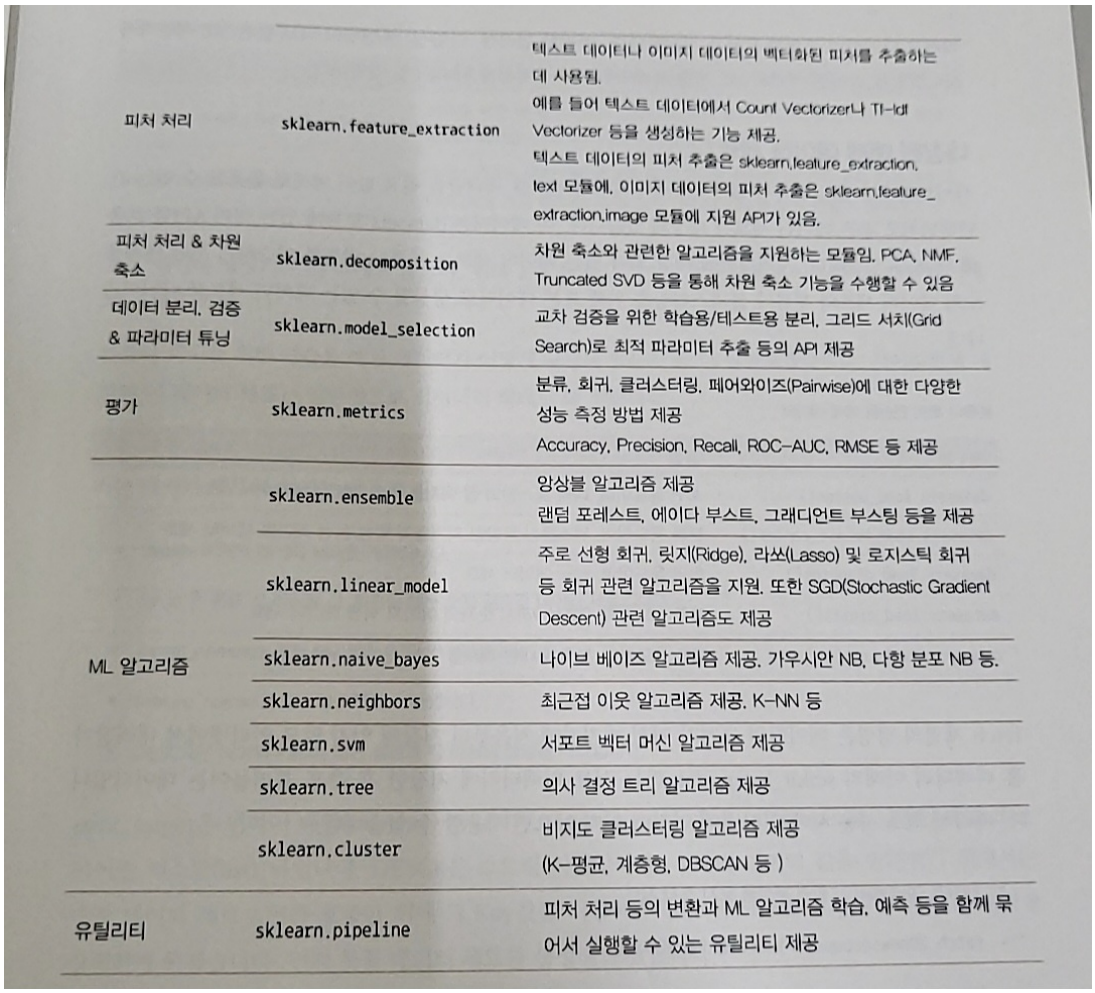
▼ 03. 사이킷런의 기반 프레임워크 익히기

- Estimator 이해 및 fit(), predict() 메서드
    - fit() : ML 모델 학습을 위함
    - predict() : 학습된 모델 예측
- 분류 알고리즘을 구현한 클래스 : Classifier
- 회귀 알고리즘을 구현한 클래스 : Regressor
- Classifier와 Regressor을 합쳐 Estimator 클래스라고 부름
    - 지도 학습 모든 알고리즘을 구현한 클래스를 통칭해 Estimator라고 부름
- cross_val_score()와 같은 evaluation 함수, 하이퍼 파라미터 튜닝을 지원하는 클래스의 경우 이 Estimator를 인자로 받음
- 비지도 학습을 구현한 클래스 역시 대부분 fit()과 transform() 적용
    - fit() : 변환을 위한 사전 구조를 맞춤
    - transform() : 데이터의 차원 변환, 클러스터링, 피처 추출 등 실제 작업 수행
    - 하나로 합친 fit_transform()도 함께 제공

- 사이킷런의 주요 모듈



분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자 형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업을 수행하는 다양한 기능 제공



피처 처리	sklearn.feature_extraction	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는 데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 TF-Idf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 sklearn.feature_extraction.text 모듈에, 이미지 데이터의 피처 추출은 sklearn.feature_extraction.image 모듈에 지원 API가 있음.
피처 처리 & 차원 축소	sklearn.decomposition	차원 축소와 관련한 알고리즘을 지원하는 모듈임. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있음
데이터 분리, 검증 & 파라미터 튜닝	sklearn.model_selection	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	sklearn.metrics	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	sklearn.ensemble	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공
	sklearn.linear_model	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
	sklearn.naive_bayes	나이브 베이즈 알고리즘 제공. 가우시안 NB, 다항 분포 NB 등.
	sklearn.neighbors	최근접 이웃 알고리즘 제공. K-NN 등
	sklearn.svm	서포트 벡터 머신 알고리즘 제공
	sklearn.tree	의사 결정 트리 알고리즘 제공
	sklearn.cluster	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
유틸리티	sklearn.pipeline	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

  - 일반적으로 머신러닝 모델을 구축하는 주요 프로세스는 피처의 가공, 변경, 추출을 수행하는 피처 처리, ML 알고리즘 학습/예측 수행, 모델 평가의 단계를 반복적으로 수행
- 내장된 예제 데이터 세트
  - 분류나 회귀 연습용 예제 데이터

분류나 회귀 연습용 예제 데이터

API 명	설명
datasets.load_boston()	회귀 용도이며, 미국 보스턴의 집 피처들과 가격에 대한 데이터 세트
datasets.load_breast_cancer()	분류 용도이며, 위스콘신 유방암 피처들과 악성/음성 레이블 데이터 세트
datasets.load_diabetes()	회귀 용도이며, 당뇨 데이터 세트
datasets.load_digits()	분류 용도이며, 0에서 9까지 숫자의 이미지 픽셀 데이터 세트
datasets.load_iris()	분류 용도이며, 붓꽃에 대한 피처를 가진 데이터 세트

- fetch 계열 명령은 최초 사용 시에 인터넷에 연결돼 있지 않으면 사용 불가
- 분류와 클러스터링을 위한 표본 데이터 생성기

분류와 클러스터링을 위한 표본 데이터 생성기

API 명	설명
datasets.make_classifications( )	분류를 위한 데이터 세트를 만듭니다. 특히 높은 상관도, 불필요한 속성 등의 노이즈 효과를 위한 데이터를 무작위로 생성해 줍니다.
datasets.make_blobs( )	클러스터링을 위한 데이터 세트를 무작위로 생성해 줍니다. 군집 지정 개수에 따라 여러 가지 클러스터링을 위한 데이터 세트를 쉽게 만들어 줍니다.

- 사이킷런에 내장된 데이터 세트는 일반적으로 딕셔너리 형태로 돼 있음
- 키는 보통 data, target, target_name, feature_names, DESCR로 구성됨
  - data : 피처의 데이터 세트
  - target : 분류 시 레이블 값, 회귀일 때는 숫자 결괏값 데이터 세트
  - target_names : 개별 레이블의 이름
  - feature_names 피처의 이름
  - DESCR : 데이터 세트에 대한 설명과 각 피처의 설명
- data, target : 넘파이 배열 타입
- target_names, feature_names는 넘파이 배열 또는 파이썬 리스트 타입
- DESCR : 스트링 타입
- 피처의 데이터 값을 반환받기 위해서는 내장 데이터 세트 API를 호출한 뒤 그 Key값을 지정하면 됨
- 데이터 키는 피처들의 데이터 값을 가리킴
  - 피처 데이터 값을 추출하기 위해서는 데이터세트.data를 이용하면 됨

▼ 04. Model Selection 모듈 소개

- 학습 데이터와 테스트 데이터 세트를 분리하거나 교차 검증 분할 및 평가, Estimator의 하이퍼 파리미터를 튜닝하기 위한 다양한 함수와 클래스 제공
- 학습/테스트 데이터 세트 분리 : train_test_split()
  - 원본 데이터 세트에서 학습 및 테스트 데이터 세트를 쉽게 분리할 수 있음
  - 첫 번째 파라미터로 피처 데이터 세트, 두 번째 파라미터로 레이블 데이터 세트를 입력 받음, 선택적으로 다음 파라미터를 입력 받음
    - test_size
    - train_size
    - shuffle
    - random_state
- 교차 검증
  - 학습 데이터와 별도의 테스트용 데이터 → 과적합에 취약함
    - 과적합은 모델이 학습 데이터에만 과도하게 최적화되어, 실제 예측을 다른 데이터로 수행할 경우에는 예측 성능이 과도하게 떨어지는 것
    - 고정된 학습 데이터와 테스트 데이터로 평가 → 테스트 데이터에만 최적의 성능을 발휘할 수 있도록 편향되게 모델을 유도
  - 문제 개선을 위해 교차 검증을 이용
  - 별도의 여러 세트로 구성된 학습 데이터 세트와 검증 데이터 세트에서 학습과 평가를 수행
  - 대부분의 ML 모델 성능 평가는 교차 검증 기반으로 1차 평가를 한 뒤 최종적으로 테스트 데이터 세트에 적용해 평가하는 프로세스
  - K 폴드 교차 검증
    - 가장 보편적으로 사용되는 교차 검증 기법
    - K개의 데이터 폴드 세트를 만들어 K번만큼 각 폴드 세트에 학습과 검증 평가를 반복적으로 수행
    - 교차 검증 시마다 검증 세트의 인덱스가 달라짐
  - Stratified K 폴드
    - 불균형한 분포도를 가진 레이블 데이터 집합을 위한 K 폴드 방식
    - 특정 레이블 값이 특이하게 많거나 매우 적어서 값의 분포가 한쪽으로 치우치는 것

        - 원본 데이터의 레이블 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습과 검증 데이터 세트를 분배
        - 일반적으로 분류에서는 이 방법으로 분할되어야 함
        - 회귀에서는 지원되지 않음
    - cross_val_score()
        - 교차 검증을 편리하게 해 줌
        - 선언 형태
            - cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')
            - estimator는 분류 또는 회귀 클래스를 의미
            - X는 피처 데이터 세트, y는 레이블 데이터 세트, scoring은 예측 성능 평가 지표를 기술
            - cv는 교차 검증 폴드 수
            - classifier가 입력되면 Stratified K 폴드 방식으로, 회귀인 경우 K 폴드 방식으로 분할
            - 반환 값은 배열 형태
- GridSearchCV : 교차 검증과 최적 하이퍼 파라미터 튜닝을 한 번에
    - 교차 검증을 기반으로 하이퍼 파리미터의 최적 값을 찾게 해줌
    - 수행 시간이 상대적으로 오래 걸림
    - fit()을 수행하면 최고 성능을 나타낸 하이퍼 파라미터의 값과 그때의 평가 결과 값이 각각 best_params_, best_score_ 속성에 기록됨

▼ 05. 데이터 전처리

- ML 알고리즘을 적용하기 전에 데이터에 대해 미리 처리해야 할 기본 사항이 있음
    - 결손값(NaN, Null 값)은 허용되지 않음 → 고정된 다른 값으로 변환해야 함
        - Null 값이 얼마 되지 않으면? 피처의 평균값 등으로 간단히 대체
        - Null 값이 대부분이라면? 해당 피처 드롭이 좋음
    - 모든 문자열 값은 숫자 형으로 변환해야 함

        - 카테고리형 피처는 코드 값으로 표현, 텍스트형 피처는 피처 벡터화 또는 불필요한 피처 삭제
- 데이터 인코딩
    - 레이블 인코딩
        - 카테고리 피처를 코드형 숫자 값으로 변환
        - 01, 02 등은 문자열이기 때문에 1,2와 같은 숫자형 값으로 변환해야 함
        - LabelEncoder 클래스로 구현
            - LabelEncoder를 객체로 생성한 후 fit()과 transform()을 호출해 레이블 인코딩 수행
        - classes_ 속성값으로 원본 확인 가능
        - classes_ 속성은 순서대로 변환된 인코딩 값에 대한 원본값을 가짐
        - 디코딩 : inverse_transform()을 통해 가능
        - 선형 회귀와 같은 ML 알고리즘에는 적용하지 않아야 함(숫자 가중치 부여 등의 문제 발생)
    - 원-핫 인코딩
        - 피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0을 표시하는 방식
        - OneHotEncoder 클래스로 변환 가능
            - 입력값으로 2차원 데이터가 필요, 희소 행렬 형태이므로 toarray()를 이용해 밀집 행렬로 변환해야 함
- 피처 스케일링과 정규화
    - 피처 스케일링 : 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업
        - 대표적인 예 : 표준화, 정규화
        - 표준화 : 평균이 0, 분산이 1인 가우시안 정규 분포를 가진 값으로 변환하는 것
            - StandardScaler : 표준화를 쉽게 지원하기 위한 클래스
        - 정규화 : 서로 다른 피처의 크기를 통일하기 위해 크기를 변환해주는 개념
            - 값을 모두 최소 0~ 최대 1의 값으로 변환
            - MinMaxScaler : 데이터 값을 0과 1 사이의 범위값으로 변환

- 학습 데이터와 테스트 데이터의 스케일링 변환 시 유의점
    - 일반적으로 fit()은 데이터 변환을 위한 기준 정보를 적용, transform()은 이렇게 설정된 정보를 이용해 데이터를 변환
    - fit_transform은 둘을 한번에 적용
    - Scaler 객체를 이용해 학습 데이터 세트로 fit()과 transform()을 적용하면 테스트 데이터 세트로는 다시 fit()을 수행하지 않고 학습 데이터 세트로 fit()을 수행한 결과를 이용해 transform() 변환을 적용해야 함
    - 즉, 학습 데이터로 fit()이 적용된 스케일링 기준 정보를 그대로 테스트 데이터에 적용해야 함
    - fit_transform()의 경우 테스트 데이터에서는 절대 사용해서는 안 됨
    - 먼저 전체 데이터 세트에 스케일링을 적용한 뒤 학습과 테스트 데이터 세트로 분리하는 것이 바람직

▼ 06. 사이킷런으로 수행하는 타이타닉 생존자 예측

실습 위주라 정리할 내용이 없으므로 생략

CH3. 평가

▼ 01. 정확도

- 실제 데이터에서 예측 데이터가 얼마나 같은지 판단하는 지표
- 정확도 = 예측 결과가 동일한 데이터 건수 / 전체 예측 데이터 건수
- 이진 분류의 경우 데이터의 구성에 따라 정확도 지표가 ML 모델 성능을 왜곡할 수 있음
    - 조건 하나만을 가지고 결정하는 별거 아닌 알고리즘도 높은 정확도를 나타내는 상황 발생
- 특히 불균형한 레이블 값 분포에서 ML 모델의 성능을 판단할 경우, 적합한 평가 지표가 아님
- 따라서 여러 분류 지표와 함께 적용해야 함

▼ 02. 오차 행렬

- 이진 분류에서 성능 지표로 잘 활용됨
- 학습된 분류 모델이 예측을 수행하면서 얼마나 헷갈리고 있는지도 함께 보여주는 지표

- 이진 분포의 예측 오류가 얼마인지와 더불어 어떠한 유형의 예측 오류가 발생하고 있는지 함께 나타냄
- TN, FP, FN, TP 형태로 나타냄
    - 예측 클래스와 실재 클래스의 Positive 결정 값(값 1)과 Negative 결정 값(값 0)의 결함에 따라 결정됨
- 사이킷런은 오차 행렬을 구하기 위해 confusion_matrix() API 제공
- 정확도 = (TN+TP)/(TN+FP+FN+TP)
- 불균형한 이진 분류 데이터 세트에서는 Positive 데이터 건수가 매우 작기 때문에 데이터에 기반한 ML 알고리즘은 Positive보다는 Negative로 예측 정확도가 높아짐 → 수치적 판단 오류 일으킴

▼ 03. 정밀도와 재현율

- Positive 데이터 세트의 예측 성능에 좀 더 초점을 맞춘 평가 지표
- 서로 보완적인 지표로 분류의 성능을 평가하는 데 적용됨
- 정밀도 = TP / (FP+TP)
    - 예측을 Positive로 한 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율
    - 양성 예측도라고도 불림
    - 실제 Negative 음성인 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하게 되는 경우 중요
        - FP를 낮추는 데 초점을 맞춤
    - precision_score() 제공
- 재현율 = TP / (FN+TP)
    - 실제 값이 Positive인 대상 중에 예측과 실제 값이 Positive로 일치한 데이터의 비율
    - 민감도, TPR라고도 불림
    - 실제 Positive 양성 데이터를 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 중요
        - FN을 낮추는 데 초점을 맞춤
        - 예) 암 판단 모델 : 양성을 음성으로 잘못 판단했을 경우 대가가 굉장히 심각

    - recall_score() 제공
- 정밀도/재현율 트레이드오프
    - 분류의 결정 임곗값을 조정해 정밀도 또는 재현율의 수치를 높일 수 있음
    - 하지만 둘은 보완적인 평가 지표이기 때문에 어느 한쪽을 높이면 다른 한 쪽의 수치는 떨어지기 쉬움
    - 사이킷런의 분류 알고리즘은 예측 데이터가 특정 레이블에 속하는지 계산하기 위해 먼저 개별 레이블별로 결정 확률을 구함
    - 개별 데이터별로 예측 확률을 반환하는 메서드인 predict_proba() 제공
        - 학습이 완료된 사이킷런 Classifier 객체에서 호출이 가능하며 테스트 피처 데이터 세트를 파라미터로 입력해주면 테스트 피처 레코드의 개별 클래스 예측 확률을 반환함
    - 임계값을 낮추면 재현율이 올라가고 정밀도가 떨어짐 : Positive 예측값이 많아지면 상대적으로 재현율 값이 높아짐
    - precision_recall_curve() : 정밀도와 재현율의 임곗값에 따른 값 변화를 곡선 형태의 그래프로 시각화하는 데 이용 가능
- 정밀도와 재현율의 맹점
    - 상호 보완할 수 있는 수준에서 적용돼야 함

▼ 04. F1 스코어

- 정밀도와 재현율을 결합한 지표
- 둘 중 어느 한 쪽으로 치우치지 않는 수치를 나타낼 때 상대적으로 높은 값을 가짐
- f1_score() API 제공

▼ 05. ROC 곡선과 AUC

- 이진 분류의 예측 성능 측정에서 중요하게 사용됨
- ROC 곡선은 수신자 판단 곡선, 머신러닝 이진 분류 모델 예측 성능 판단하는 중요한 평가 지표
    - FPR이 변할 때 TPR이 어떻게 변하는지를 나타내는 곡선
    - TRR은 재현율, TNR은 민감도에 대응하는 지표인 특이성
    - FPR = FP / (FP+TN) = 1 - TNR = 1 - 특이성
    - ROC 곡선은 직선에 가까울수록 성능이 떨어지는 것

    - roc_curve() API 제공
  - AUC 값은 ROC 곡선 밑의 면적을 구한 것으로서 일반적으로 1에 가까울수록 좋은 수치
  - 보통의 분류는 0.5 이상의 AUC를 가짐

▼ 06. 피마 인디언 당뇨병 예측

실습 과정으로 이론 생략