📘 1.1, 1.2 머신러닝의 개념

- 데이터를 기반으로 한 패턴학습, 결과예측 알고리즘
- 기존에 존재하던 복잡한 소스코드들의 문제를 해결
- 지도학습(분류, 회귀, 추천 시스템, 시각/음성 감지/인지, 텍스트 분석 NLP), 비지도학습(클러스터링, 차원 축소, 강화학습)
- 데이터에 매우 의존적 ->좋은 품질의 데이터를 갖추는 것이 중요

📘 1.3 Numpy

- 선형대수, 통계 기반. 루프 없이 대량 데이터의 배열 연산 가능
- 판다스의 데이터프레임보단 편리성에서 밀림
- 기반 데이터 타입 : ndarray

numpy 기본함수

- `array()` -> 입력받은 리스트를 array로 바꿈
- `ndarray.tolist()` -> 입력받은 array를 리스트로 바꿈
- `ndarray.shape` -> 입력받은 array의 형태
- `ndarray.ndim` -> 입력받은 array의 차원
- `ndarray.dtype` -> ndarray 내의 데이터타입
- `arange(start = 0, stop)` -> start부터 stop-1 까지의 값을 1차원 array로 만들어줌
- `zeros((shape), dtype = 'float64')` -> 모든 원소가 0인 행렬을 입력받은 형태에 맞춰 반환
- `ones((shape), dtype = 'float64')` -> 모든 원소가 1인 행렬을 입력받은 형태에 맞춰 반환
- `ndarray.reshape(shape)` -> 입력받은 형태에 맞춰 행렬 바꿈
    - ndarray.reshape(-1, n) -> n열을 갖게끔 알맞은 행 개수로 맞춤
    - ndarray.reshape(n, -1) -> n열을 갖게끔 알맞은 열 개수로 맞춤
    - 지정된 사이즈로 변경 불가시 오류발생
    - ndarray.reshape(-1, 1) -> 반드시 1개 칼럼을 가진 2차원 ndarray로 변환 -> 형태 통일에 유용

numpy 인덱싱

특정 데이터 추출

- ndarray[n] -> array의 n+1번째 데이터 추출

슬라이싱

- ndarray[n:] -> array의 n+1번째 데이터 추출

- 리스트의 슬라이싱과 동일.
- 2차원 이상의 경우 [a:b, c:d] 이런 형식으로 추출 가능

팬시 인덱싱

- 리스트, ndarray로 인덱스 집합 지정 -> 해당 위치 데이터 반환

불린 인덱싱

- ndarray[조건] -> 해당 조건에 해당하는 ndarray 반환
- [조건]이 곧 True, False로 구성된 array가 됨. 이때 ndarray[조건]는 True값을 가지는 인덱스를 저장, 그에 대한 데이터로 ndarray를 조회

행렬 정렬

- `np.sort()` -> 기존행렬 유지, 정렬된 행렬 반환
  - `np.sort()[::-1]` -> 내림차순 정렬, 원래 기본은 오름차순 정렬.
  - axis=0 -> 로우방향정렬(↓)
  - axis=1 -> 칼럼방향정렬(→)
- `ndarray.sort()` -> 기존행렬 정렬, 반환값 없음
- `np.argsort()` -> 원본행렬이 정렬되었을 때 원본행렬에 대한 인덱스 반환
  - org = np.array([3, 1, 9, 5]) -> np.argsort(org) = [1, 0, 3, 2]

행렬 연산

- `np.dot(A, B)` -> 행렬 내적
- `np.transpose(A)` -> 전치 행렬

📘 1.4 Pandas

- DataFrame -> 2차원 데이터 구조체
- Index -> 데이터 프레임의 개별 데이터를 식별하는 고유 key값
- Series -> 칼럼이 하나인 데이터 구조체

*** `pd.read_csv(filepath, sep = ',',...)` 으로 외부 데이터를 데이터프레임으로 불러오기 가능. 구분문자는 콤마가 디폴트 ***

판다스 기본 함수

`df.head()` -> 가장 앞의 행 5개(디폴트) 보여줌 `df.shape` -> 데이터프레임이 몇 행 몇 열인지 반환 `df.info()` -> 데이터프레임의 데이터 타입 알려줌 `df.describe()` -> 데이터프레임의 칼럼별 통계량(평균, 행 개수, 표준편차, 최대최소, Quatile 등) `df[칼럼명].value_counts()` -> 칼럼값의 유형, 유형별 건수

- dropna = True가 디폴트 -> nul값도 카운트에 포함시키기 위해선 False로 바꾸고 실행.

ndarray, list, dict -> DataFrame

- `pd.DataFrame(list or ndarray, columns = 칼럼이름리스트)`
- `pd.DataFrame(dict)` -> 딕셔너리의 키값이 칼럼명, value가 칼럼 데이터로 매핑.

DataFrame -> ndarray, list, dict

- `df.values` -> ndarray
- `df.values.tolist` -> list
- `df.to_dict('list')` -> dict(근데 이제 value가 리스트인.)

칼럼 데이터 세트 생성, 수정, 삭제

- `df[새 칼럼명] = new_value` -> 새롭게 생성된 칼럼의 모든 데이터가 new_value로 변경
- `df[새 칼럼명] = f(df[기존 칼럼명])` -> 새롭게 생성된 칼럼에 기존 칼럼 시리즈의 함수값이 데이터로 들어감
- `df.drop(칼럼명/로우인덱스, axis = 0, inplace = False)`
    - 삭제할 칼럼/로우가 여러 개면 리스트 형식으로 입력
    - axis = 0 -> 행 삭제 axis = 1 -> 열 삭제
    - inplace = False -> 삭제된 데이터 프레임 반환, but 기존 데이터프레임은 변화 x
    - inplace = True -> 기존 데이터 프레임에서 행/열 삭제, but 바뀐 데이터프레임은 반환 x

인덱스 객체

- `df.index` -> RangeIndex(start, stop, step)
    - 1차원 array로 데이터프레임의 인덱스를 보여줌
    - 인덱스 객체는 수정이 불가
    - 인덱스 개체는 연산에서 제외, 오직 식별에만 사용. 만약 인덱스 연산이 필요하다면 `df.reset_index()`로 index라는 새로운 칼럼을 df에 추가하는 방법 사용.
- `df.reset_index()` -> 인덱스가 연속 정수형으로 만드는 경우에 사용.
    - `df.reset_index(drop = True)` -> index라는 새로운 칼럼 생성 x
    - 위에서 drop = False(디폴트)이면 index라는 새로운 칼럼 생성, 따라서 이 경우 Series는 DataFrame으로 바뀜

데이터 셀렉션, 필터링

- `df[]` -> 칼럼 지정 연산자(칼럼명 지정, 불린 인덱싱)
- `df.iloc[행 위치 정수값, 열 위치 정수값]` -> 위치 기반, 따라서 불린 인덱싱 제공 안됨.
- `df.loc[행 인덱스 이름, 열 이름]` -> 명칭 기반
    - 행 인덱스는 0 -> 'one' 1 ->'two' 이렇게 문자열로 표현해야함.

- 불린 인덱싱
    - `df[조건][[보고 싶은 칼럼명 나열]]`
    - `df.loc[조건, [보고 싶은 칼럼명 나열]]`
    - 조건을 변수에(con1, con2, ...) 저장하고 con1&con2&... 과 같이 결합해서 조건 자리에 넣어 불린 인덱싱 가능

정렬 , Aggregation, GroupBy

- `df.sort_values()` -> 입력값을 기준으로 오름파순 정렬
    - ascending = False -> 내림차순정렬
    - inplace = True -> 기존 데이터 프레임 변화, 리턴값 없음
    - inplace = False -> 기존 데이터 프레임 변화x, 리턴값 있음
    - by = [칼럼명 나열] -> 해당 옵션 추가시 여러 칼럼을 기준으로 정렬 가능
- `df[[칼럼명 나열]].aggregation()` -> 해당 칼럼들에 대해서 연산 가능. 평균, 최대 최소, 총합 등등
    - `df[[칼럼명 나열]].agg([aggregation 함수 나열])` -> 해당 데이터에 대해 여러 연산 한번에 가능
- `df.groupby(by = '칼럼명')` -> 해당 칼럼값을 기준으로 group화됨

결손 데이터 처리

- `df.isna()` -> Null 값이면 해당 데이터에 True, 아니면 False를 반환
- `df.fillna('new_null')` -> Null 값을 입력받은 값으로 바꿈

apply lambda 식의 데이터 가공

- lambda x : f(x) 와 같이 함수를 쓸 수 있다.
- df.apply(lambda x : f(x)) -> 해당 데이터 값(x)에 대한 f(x) 데이터를 반환
    - `titanic_df['Age'].apply(lambda x : 'Child' if x<=15 else ('Adult' if x<= 60 else 'Elderly'))` 해당 예시처럼 조건문도 가능

📘 2.1 사이킷런 소개

- 가장 대표적인 머신러닝 라이브러리

📘 2.2 2.3 사이킷런의 기반 프레임워크 익히기

내장 데이터 세트

- iris, Boston, diabetes 등등

- 데이터가 주로 딕셔너리 형태이므로 data.key로 데이터의 특성을 알 수 있다.
    - data.data -> 피처의 데이터 세트
    - data.target -> 분류시 레이블의 값
    - data.target_names -> 개별 레이블 이름
    - data.feature_names -> 피처의 이름
    - data.DESCR -> 데이터 세트 설명, 피처 설명

📘 2.4 Model Selection

train_test_split()

```
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size = 0.25)
```

- 학습 데이터와 테스트 데이터를 분리하지 않고 모두 학습 데이터에 이용해 버리면 언제나 예측 정확도가 1이 나온다. \n 그야 당연히 그걸로 만든건데 그걸로 또 테스트를 하면 당연히 그에 맞는 값이 나온다... 그래서 데이터를 분리하는 것이 중요하다. 해당 모듈이 그런 역할!
- test_size는 디폴트가 0.25로 테스트 데이터가 전체 데이터의 25%라는 뜻.
- shuffle =True가 디폴트, 섞어야 좀 더 랜덤하게 데이터 분류가 가능함.
- random_state는 seed처럼 사용되는 것, 항상 일관된 랜덤값을 준다

교차 검증

- 특정 데이터로 모델을 학습시킬 때 해당 모델이 해당 데이터에만 과도하고 최적화되는 경우 다른 데이터엔 적용이 어려울 수 있다. \n 이를 방지하기 위해 데이터를 여러번 나눠서 여러번 검증을 하는 교차 검증이 사용된다.

KFold

- 데이터를 K 등분 후 K번째 데이터를 검증세트로 해서 K번 학습/교차 검증

Stratified KFold

- 보통의 Kfold는 label값의 비율이 잘 맞춰지지 않아서 편향된 학습이 될 가능성 존재, 따라서 해당 비율을 일정하게 맞춰주는 과정을 추가

cross_val_score()

- cross_val_score(estimator, X, y, scoring, cv)
    - 차례대로 알고리즘 클래스, 피처 데이터 세트, 레이블 데이터세트, 예측성능평가지표(ex. 정확도), 교차검증 폴드 횟수
    - 내부적으로 Stratified KFold가 사용되어서 간단하게 Stratified KFold 가능.

GridSerachCV

- 교차 검증을 기반으로 해서 하이퍼 파라미터의 최적 값을 찾아냄, 기술된 모든 파라미터를 순차적으로 적용해서 최적의 파라미터를 찾아냄.

- 그래서 시간 오래걸림

📘 2.5 데이터 전처리

데이터 인코딩

- 레이블 인코딩
  - LabelEncoder() -> 각각의 레이블 값을 숫자 값으로 변환해줌(단 가중치 아니고 그냥 단순 코드임)
- 원-핫 인코딩
  - 피처 값을 아예 새로운 칼럼으로 만들어버리고 그에 해당하면 1, 아니면 0 을 반환하는 인코딩
  - OneHotEncoder()
  - `pd.get_dummies(df)` 로 보다 더 간단히 적용 가능

피처 스케일링과 정규화

- 표준화 : 데이터가 평균=0 분산=1인 정규분포에 맞게 스케일링
  - StandardScaler로 객체 생성 후 fit, transform 이용해서 스케일링
- 정규화 : 단위가 다른 변수를 모두 동일한 크기 단위로 비교하기 위해 값을 모두 0~1 사이로 스케일링
  - MinMaxScaler
- 학습 데이터와 테스트 데이터 스케일링 변환 시 유의점.
  - 학습 데이터와 테스트 데이터의 최대/최솟값이 달라 스케일링을 따로 하면 해당 데이터의 스케일링이 엇나갈 수 있다.
  - 따라서 가능하다면 전체 데이터를 스케일링한 이후에 데이터를 나누는 것이 좋다.
  - 그래도 안된다면 학습데이터로 스케일러 객체를 fit 시킨 이후에 그를 테스트 데이터에 적용하고 transform.

📘 3.1 정확도

- 정확도 = 예측 결과가 맞는 데이터 건수/전체 예측 데이터 건수
  - 불균형한 레이블 분포에서 사용할 시엔 적합하지 않을 수 있다. (답이 1인게 2%만큼 있는데 다 답 아니라고 하면 정확도는 98%니까..)

📘 3.2 오차 행렬

- TN(N 예측, 맞음). FP(P 예측, 틀림). FN(N 예측, 틀림), TP(P 예측, 맞음) 로 이루어진 행렬 각각 1, 2, 3, 4분면
- 정확도 = (TN+TP)/(TN+FP+FN+TP) 로 나타내기 가능

📘 3.3 정밀도와 재현율

- 정밀도 = TP /(FP+TP)
    - 음성 데이터를 양성 데이터로 판단하면 큰일 나는 경우에 중요
    - FP를 낮추는 데 초점
- 재현율 = TP /(FN+TP)
    - 양성 데이터를 음성 데이터로 판단하면 큰일 나는 경우에 중요(암 진단, 금융사기 판단 등)
    - FN을 낮추는 데 초점
- 트레이드 오프
    - 정밀도와 재현율은 상호보완적 -> 하나가 올라가면 하나가 떨어지기 마련.(=트레이드 오프)
- 맹점
    - 둘 중 어느 하나만 높아서는 좋은 평가라고 할 수 없다.

📘 3.4 F1 Score

- 위의 정밀도와 재현율의 맹점을 보완.
- 둘 중 어느 한 쪽으로 치우치지 않는 수치를 나타낼 때 상대적으로 높은 값을 가짐. (밸런스 중시)

📘 3.5 ROC곡선과 AUC

- TPR(True positive Rate)
- TNR(True Negative Rate)
- FPR(False positive Rate)
- FPR = 1- TNR
- ROC 곡선: FPR이 변할 때 TPR이 변하는 모습을 곡선으로 나타냄
    - 따라서 해당 곡선이 가운데 직선에서 멀어질수록, 즉 가장 모서리에 가까워질 수록 더 성능이 뛰어나다고 판단.
    - ->AUC = ROC 곡선 밑의 면적으로 넓을 수록 더 좋은 평가
- ROC-AUC: 이진 분류의 성능 평가를 위해 가장 많이 사용