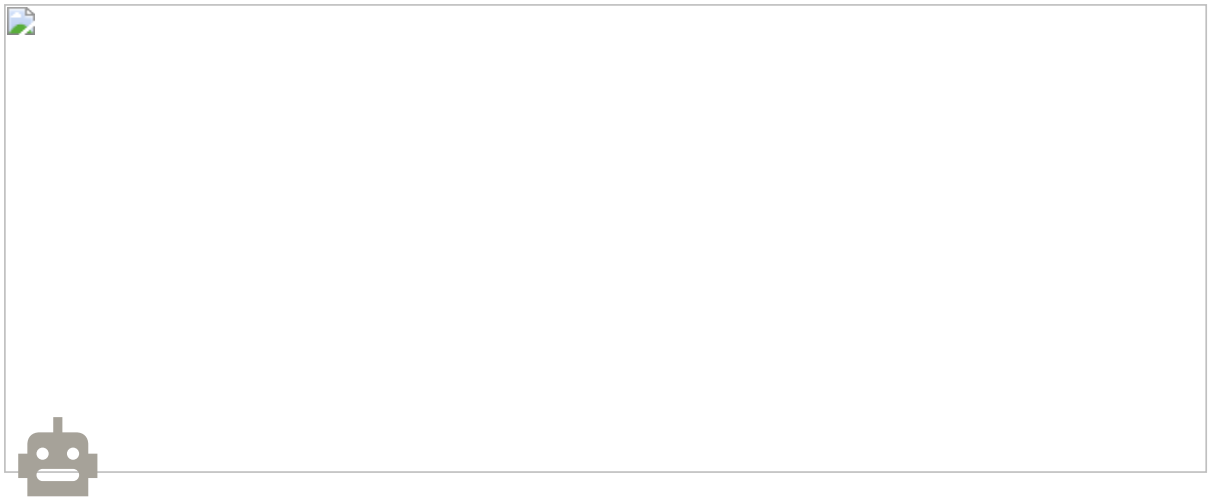

1-3장

1️⃣ 파이썬 기반의 머신러닝과 생태계 이해

3) 넘파이

3-1. 개요

3-2. ndarray의 데이터 타입 arange, zeros, ones

- ndarray 내의 데이터 값은 숫자, 문자열, bool 등 모두 가능
- ndarray 내의 데이터 타입은 같은 데이터 타입만 가능
- `dtype` 속성으로 ndarray 내의 데이터 타입 확인 가능
- `astype()` 함수를 통해 ndarray 내의 데이터 값의 타입 변경
  - astype(데이터형)
  - 대용량 데이터의 ndarray 생성 시 메모리 절약을 위해 보통 이용됨

3-3. arange, zeros, ones

- `arange(start = 0, stop, stap)` : start부터 (stop-1)까지의 값을 순차적으로 ndarray의 데이터 값으로 반환
- `zero((nrows, ncols))` : 튜플 형태의 shape 값 입력 시 모든 값을 0으로 채운 해당 shape를 가진 ndarray 반환
- `ones((nrows, ncols), dtype = float64)` : 튜플 형태의 shape 값 입력 시 모든 값을 1으로 채운 해당 shape를 가진 ndarray 반환

3-4. reshape()

- ndarray를 특정 차원과 크기로 변환
- -1을 인자로 사용하면 원래 ndarray와 호환되는 새로운 shape으로 변환해준다.

ex) `array1.reshape(-1, 5)` : 5개의 열을 가진 행렬로 자동화

3-5. 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)

일부 데이터 세트나 특정 데이터만을 선택할 수 있도록

- 단일 값 추출 : 인덱스 -1은 맨 뒤의 데이터
  - `axis0` : 로우 방향 축, `axis1` : 칼럼 방향 축
- 슬라이싱
- 팬시 인덱싱 : list나 ndarray로 인덱스 집합을 지정하면 해당위치의 인덱스에 해당하는 ndarray를 반환
- 불린 인덱싱

3-6. 행렬의 정렬 - sort()와 argsort()

- 행렬 정렬
  - `np.sort()` : 원 행렬은 그대로 유지, 정렬된 행렬 반환
  - `ndarray.sort()` : 원 행렬 자체를 정렬한 형태로 변환, 반환 값은 None
  - 내림차순 반환은 ex) `np.sort()[::-1]`
  - 2차원 이상 행렬
    1. `np.sort(A, axis=0)` : 로우 방향으로 정렬
    2. `np.sort(A, axis=1)` : 칼럼 방향으로 정렬
  - `np.argsort()` : 정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 반환

3-7. 선형대수 연산 - 행렬 내적과 전치 행렬 구하기

- 행렬 내적(행렬 곱)
  - `np.dot()`
- 전치 행렬
  - `transpose()`

4) 판다스

- kaggle에서 csv 파일 내려받기
- * 코랩 기준 파일 불러오기

```
!pip install kaggle  # Kaggle API 패키지 설치

# Kaggle API 토큰 업로드 (kaggle.json 파일을 업로드해야 함)
from google.colab import files
files.upload()

# Kaggle API 토큰을 알맞은 위치로 이동하고 권한 부여
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Kaggle 데이터셋 다운로드 (아래 your_dataset_name는 Kaggle API 명령어를 붙여넣기)
!kaggle datasets download -d your_dataset_name
```

```
import pandas as pd

# CSV 파일 읽기 (파일 이름은 다운로드한 파일의 이름으로 바꾸기)
df = pd.read_csv('your_downloaded_csv_file.csv')

# DataFrame을 사용하여 데이터 분석 또는 처리
# 예: 데이터의 처음 몇 개 행 출력
print(df.head())

#print(df.head(3)) 이렇게 입력하면 처음 세 개의 행만 출력된다 !
```

- df.info() : 총 데이터 건수와 데이터 타입, null 건수
- describe() : 칼럼별 숫자형 데이터값(int, float)의 n-percentile 분포도, 평균값, 최댓값, 최솟값

4-1. 기본 API

- `value_counts()` : null 값을 무시하고 결괏값을 내놓기 쉬움
  - `dropna` 인자로 판단 ; 기본값 true, null 값을 무시하고 개별 데이터 값의 건수를 계산

    null 값 포함하여 계산 진행하고 싶다면 ex) `value_counts(dropna=False)`

4-2. DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

- 2차원 이하의 데이터들만 DataFrame(2차원)으로 변환 가능
  - 1차원: 리스트, 넘파이 ndarray → column은 1개만 필요
- `pd.DataFrame()`
- DataFrame → 리스트 : values로 얻은 ndarray에 tolist() 호출
- DataFrame → 딕셔너리 : DataFrame 객체의 to_dict() 메서드를 호출하는데, 인자로 'list'를 입력하면 딕셔너리의 값이 리스트형으로 반환됨

4-3. DataFrame의 칼럼 데이터 세트 생성과 수정

- 생성

  ex) `titanic_df['Age_0']=0` : Titanic DataFrame의 새로운 칼럼 Age_0을 추가하고 일괄적으로 0할당

4-4. DataFrame 데이터 삭제

*page 58

- `drop()`
- `inplace=False` 가 기본값인데 이는 자기 자신의 DataFrame의 데이터는 삭제하지 않으며, 삭제된 결과 DataFrame을 반환 → `inplace=True`로 설정하면 자신의 DataFrame의 데이터를 삭제, 반환값은 아무 값도 아님(None)이 된다.

4-5. Index 객체

```
# index 객체 추출
indexes = titanic_df.index
print(indexes)
# index 객체를 실제 값 array로 변환
print('Index 객체 array값:\n', indexes.values)
```

- 인덱스는 오직 식별용으로 사용 (연산 x)
- `reset_index()` : 새롭게 인덱스를 연속 숫자형으로 할당하며 기존 인덱스는 'index'라는 새로운 칼럼명으로 추가한다.

4-6. 데이터 셀렉션 및 필터링

- [ ] : 넘파이에서는 행의 위치, 열의 위치, 슬라이싱 번위 등 지정

  그러나 DateFrame에서는 칼럼명 문자 또는 인덱스로 변환 가능한 표현식이 들어감

  (= 칼럼만 지정할 수 있는 *칼럼 지정 연산자*)

- `iloc[]` : 위치 기반 인덱싱 : 행과 열 위치를, 0을 출발점으로 하는 세로축, 가로축, 좌표 정숫값으로 지정하는 방식
  - 열 위치에 -1을 입력하여 데이터 프레임의 가장 마지막 열 데이터를 가져오는 데 자주 사용
  - `iloc[:, -1]` : 맨 마지막 칼럼의 값 (타깃값)
  - `iloc[:, :-1]` : 처음부터 맨 마지막 칼럼을 제외한 모든 칼럼의 값 (피처값)
- `loc[]` : 명칭 기반 인덱싱 : 데이터 프레임의 인덱스 값으로 행 위치를, 칼럼의 명칭으로 열 위치를 지정하는 방식

4-7. 정렬, Aggregation 함수, GroupBy 적용

- `sort_values()`
- `groupby()` : (by=)이 안에 칼럼을 입력하면 대상 칼럼

```
titanic_groupby = titanic_df.groupby(by='Pclass')
print(type(titanic_groupby))
```

```
<class 'pandas.core.groupby.generic.DataFrameGro
```

- 여러 칼럼이 서로 다른 aggregation 함수를 groupby에서 호출

```
agg_format={'Age':'max', 'SibSp':'sum', 'Fare':'mean'}
titanic_df.groupby('Pclass').agg(agg_format)
```

4-8. 결손 데이터 처리하기

- null 즉 NaN 여부를 확인하는 API는 `isna()`
- NaN 값을 다른 값으로 대체하는 API는 `fillna()`

4-9. apply lambda 식으로 데이터 가공

복잡한 데이터 가공이 필요할 경우

- `lambda x : cat(x)` : x를 넣으면 cat라는 함수에 적용

2️⃣ 사이킷런으로 시작하는 머신러닝

2) 첫번째 머신러닝 만들어보기 - 붓꽃 품종 예측

2-2. 붓꽃 품종 예측하기

- ML 알고리즘은 의사 결정 트리 알고리즘으로, 이를 구현한 DecisionTreeClassifier를 적용

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

import pandas as pd

# 붓꽃 데이터 세트를 로딩
iris = load_iris()

# iris.data는 iris 데이터 세트에서 피처만으로 된 데이터를 numpy로 가지고 있다.
iris_data = iris.data

# iris.target은 붓꽃 데이터 세트에서 레이블(결정 값) 데이터를 numpy로 가지고 있다.
iris_label = iris.target
print('iris target값:', iris_label)
print('iris target명:', iris.target_names)

# 붓꽃 데이터 세트를 자세히 보기 위해 DataFrame으로 변환
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)
iris_df['label'] = iris.target
iris_df.head(3)

X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label, test_size=0.2, random_state=11)

# DecisionTreeClassifier 객체 생성
dt_clf = DecisionTreeClassifier(random_state=11)

# 학습 수행
dt_clf.fit(X_train, y_train)

# 학습이 완료된 DecisionTreeClassifier 객체에서 테스트 데이터 세트로 예측 수행
pred = dt_clf.predict(X_test)

from sklearn.metrics import accuracy_score
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

3) 사이킷런의 기반 프레임워크 익히기

2-3. Estimator 이해 및 fit(), predict()

- Estimator = Classifier(분류) + Regressor(회귀)
- 학습 - `fit()`
- 예측 - `predict()`

4) Model Selection 모듈 소개

4-1. 학습/데이터 세트 분리

- `train_test_split()` : 원본 데이터 세트에서 학습/테스트 데이터 세트 분리

  학습과 예측을 동일한 데이터 세트로 수행하면 이미 학습한 데이터 세트를 기반으로 예측했기 때문에 정확도 100이 나오는 이상한 결과 → 전용의 테스트 데이터 세트여야 함

  - `test_size` : 전체 데이터에서 테스트 데이터 세트 크기를 얼마로? (dafault : 0.25)
  - `train_size` : 전체 데이터에서 학습용 데이터 세트 크기를 얼마로?
  - `shuffle` : 데이터 분리 전 데이터를 미리 섞을 것인가? (dafault=True)

- `random_state` : 일정한 숫자값 부여하면 수행할 때마다 동일한 데이터 세트로 분리

4-2. 교차검증

- k 폴드 교차 검증
  1. 폴드 세트 설정
  2. for 루프에서 반복으로 학습 및 테스트 데이터의 인덱스 추출
  3. 반복적으로 예측 수행, 예측 성능 반환
- Stratified K 폴드 : 불균형한 분포도를 가진 레이블 데이터 집합을 위한 k 폴드 방식
- `cross_val_score()` : 교차 검증을 보다 간편하게

4-3. GridSearchCV

교차 검증과 최적 하이퍼 파라미터 튜닝을 한 번에 !

- 모든 파라미터를 순차적으로 적용해 최적의 파라미터를 찾을 수 있지만 수행시간 ⬆️
- `cv` : 교차 검증을 위해 분할되는 학습/테스트 세트의 개수 저장
- `param_grid` : key + 리스트 값을 가지는 딕셔너리, 파라미터명과 사용될 여러 파라미터 값을 지정
- `fit` (학습 데이터 세트) : 학습 데이터를 cv에 기술된 폴딩 세트로 분할해 `param_grid`에 기술된 하이퍼 파라미터를 순차적으로 변경하면서 학습/평가를 수행하고 그 결과를 `cv_results_` 속성에 기록
- `cv_results_` : gridsearchcv의 결과 세트, 딕셔너리 형태로 key값과 리스트 형태의 value값

5) 데이터 전처리

5-1. 데이터 인코딩

- 레이블 인코딩 : 카테고리 피처 → 코드형 숫자
  - `LabelEncoder`를 객체로 생성 후, `fit()`과 `transform()`으로 레이블 인코딩 수행
- 원-핫 인코딩
  - `OneHotEncoder` 클래스 : 입력값으로 2차원 데이터 필요, 후에 `toarray()` 메소드 이용해 밀집행렬로 변환해야함
  - `get_dummies` : OneHotEncoder와 다르게 숫자형 변환 필요 없이 바로 변환 가능

5-2. 피처 스케일링과 정규화

서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

- **StandardScaler**
- **MinMaxScaler**

3️⃣ 평가

1) 정확도

```
y = (digits.target == 7).astype(int)
```

2) 오차 행렬

TN(예측:negative, 실제:negative)	FP(예측:positive, 실제:negative)
FN(예측:negative, 실제:positive)	TP(예측:positive, 실제:positive)

- confusion_matrix()
- 정확도 = 예측 결과와 실제 값이 동일한 건수/전체 데이터 수 = (TN + TP)/(TN + FP + FN +TP)

3) 정밀도와 재현율

불균형한 데이터 세트에서 정확도보다 더 선호되는 평가 지표

- 정밀도 = TP / (FP + TP) : `precision_score()`
- 재현율 = TP / (FN + TP) : `recall_score()`

3-1. 정밀도/재현율 트레이드오프

- `predict_proba()` : 텍스트 피처 레코드의 개별 클래스 예측 "**확률**" 반환
  - 형태 : ndarray m x n (m : 입력값의 레코드 수, n : 클래스 값 유형)
- Binarizer 클래스
  - `fit_transform()`
  - 임곗값을 낮추면 재현율 값이 올라가고 정밀도가 떨어짐(True 값 ⬆️)

4) F1 스코어

f1 스코어가 높을수록 정밀도와 재현율이 어느 한쪽으로 치우치지 않았단 뜻

- `f1_score()`

5) ROC 곡선과 AUC

이진 분류의 예측 성능 측정에서 중요한 지표

- `roc_curve()` : FPR, TPR, 임곗값이 반환값
  - 입력 파라미터 : y_true, y_score
  - 반환값 : fpr, tpr, thresholds