사이킷런, 평가

Select 1주차

2️⃣ 사이킷런

2.1 붓꽃 품종 예측하기

2.1.1 사이킷런 모듈 import

- 지도학습 방법 중 하나인 분류 (Classification)
- **학습 데이터 세트**로 모델을 학습한 뒤, 별도의 **테스트 데이터 세트**에서 미지의 label을 예측한다.
- `sklearn.datasets` : 데이터 세트 생성하는 모듈의 모임
- `sklearn.tree` : 트리기반 ML 알고리즘 구현 클래스의 모임
  - DecisionTreeClassifier : 의사 결정 트리 클래스
- `sklearn.model_selection` : 학습 데이터와 테스트 데이터 세트 분리하거나 최적의 하이퍼 파라미터로 평가하기 위한 모듈의 모임
  - 하이퍼 파라미터: 머신러닝 알고리즘별로 최적의 학습을 위해 직접 입력하는 파라미터들이다. 이를 통해 머신러닝 알고리즘의 성능을 튜닝할 수 있다.

👉 **label**
예측하고자 하는 항목이다.
e.g. Setosa → 0, Versicolor → 1, Virginica → 2

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

2.1.2 데이터 세트 로딩

```
import pandas as pd

# 붓꽃 데이터 세트를 로딩
iris = load_iris()

# iris.data는 Iris 데이터 세트에서 feature 만으로 된 데이터를 numpy로 가
iris_data = iris.data

# iris.target은 붓꽃 데이터 세트에서 레이블(결정 값) 데이터를 numpy로 가지
iris_label = iris.target

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

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

2.1.3 데이터 세트 분리

- `train_test_split(feature 데이터 세트, label 데이터 세트, test_size=, random_state = )`

  : 학습용 데이터와 테스트용 데이터 분리

  - test_size=0.2 → 전체 데이터 중 테스트 데이터가 20% , 학습 데이터가 80%
  - random_state =11 : 매번 동일한 학습 / 테스트 데이터 세트를 생성하기 위함

  ```
  # X_train: 학습용 feature 데이터 세트
  # X_test: 테스트용  feature 데이터 세트
  # y_train: 학습용  label 데이터 세트
  ```

```
# y_test: 테스트용  label 데이터 세트

X_train, X_test, y_train, y_test = train_test_split(iris_data
                                                    test_size
```

2.1.4 의사 결정 트리 이용한 학습

- DecisionTreeClassifier 객체 생성
- `fit` 메서드에 학습용 feature 데이터 속성과 결정값 데이터 세트 입력해 호출

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

# 학습 수행
dt_clf.fit(X_train, y_train)
```

2.1.5 예측

- predict 메서드에 테스트용 feature 데이터 세트 입력하면 예측값 반환해준다.

```
pred = dt_clf.predict(X_test)
```

2.1.6 예측 성능 평가

- `accurancy_score(실제 레이블 데이터 세트, 예측 레이블 데이터 세트)`
  - 예측 결과가 실제 레이블 값과 얼마나 정확하게 맞는지 정확도 측정

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

2.2 사이킷런 기반 프레임워크

2.2.1 Estimator 및 fit, predict 메서드

🦫 지도학습

- `Estimator` : 지도학습의 모든 알고리즘을 구현한 클래스를 통칭하는 말
  - `Classifier` : 분류 알고리즘을 구현한 클래스
  - `Regressor` : 회귀 알고리즘을 구현한 클래스
  - fit 과 predict 만을 이용해 간단하게 학습과 예측 결과 반환

    (`fit` : ML 모델 학습 / `predict` : 학습된 모델의 예측)

🦫 비지도학습 (차원 축소, 클러스터링) & feature 추출

- `fit` : 입력 데이터의 형태에 맞춰 데이터 변환하기 위해 사전 구조 맞추기
- `transform` : fit 이후 입력 데이터의 차원 변환, 클러스터링, feature 추출 등의 실제 작업 수행

2.2.2 표본 데이터 생성

`datasets.make_classifications` : 분류를 위한 데이터 세트 생성

`datasets.make_blobs` : 클러스터링을 위한 데이터 세트 생성

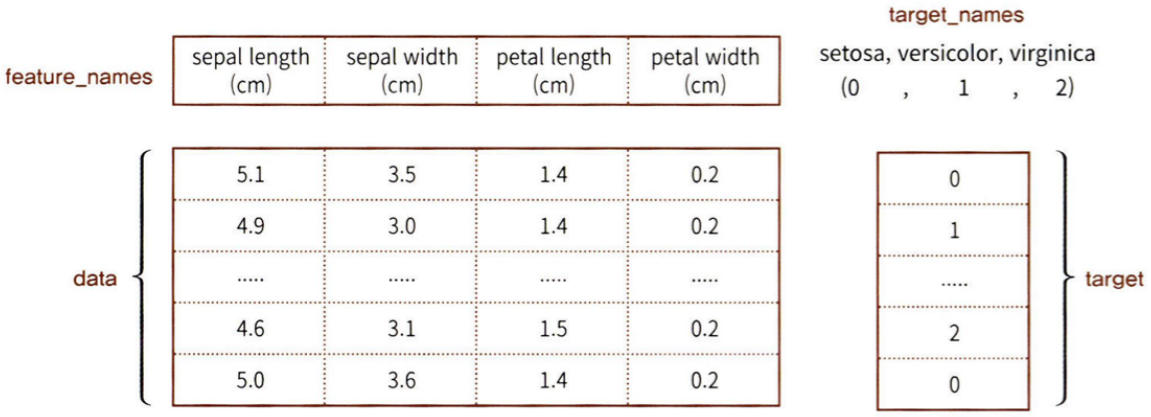
2.2.3 예제 데이터

- 딕셔너리 형태로 되어 있다. (Bunch 클래스)

키	의미	데이터 타입
data	feature의 데이터 세트	넘파이 배열 (ndarray)

target	분류 시 label 값. 회귀일 때는 숫자 결과 값	넘파이 배열 (ndarray)
target_names	개별 label의 이름	넘파이 배열 (ndarray) 또는 파이썬 리스트
feature_names	feature의 이름	넘파이 배열 (ndarray) 또는 파이썬 리스트
DESCR	데이터 세트에 대한 설명과 각 feature의 설명	스트링

- key는 feature들의 데이터 값



2.3 Model Selection

2.3.1 소개

- 학습 데이터와 테스트 데이터 분리
- 교차 검증 분할 및 평가
- Estimator의 하이퍼 파라미터 튜닝

2.3.2 학습 / 테스트 데이터 분리 → train_test_split

`train_test_split(feature 데이터 세트, label 데이터 세트, test_size=, train_size= , shuffle= ,random_state = )`

파라미터	의미

feature 데이터 세트	필수
label 데이터 세트	필수
test_size	- 전체 데이터에서 테스트 데이터 세트 크기를 얼마로 할 것인가. - 디폴트는 0.25 (25%) - 반대로 학습용 데이터의 비율을 설정하는 train_size도 있으나 사용 빈도 낮음
shuffle	- 데이터를 섞을지 여부 결정 - 디폴트는 True
random_state	호출할 때마다 동일한 학습 / 테스트 데이터 생성

2.3.3 교차 검증

과적합 (Overfitting)

모델이 학습 데이터에만 과도하게 최적화되어 실제 예측을 다른 데이터로 수행할 경우에는 성능이 떨어지는 것이다.

교차검증

과적합을 해결하기 위한 것이 교차검증이다. 학습 데이터 세트 중 일부를 별도의 검증 데이터 세트로 사용한다.

K 폴드 교차 검증

- 데이터를 k 등분 하여 그 중 하나를 검증 세트로, 나머지를 학습 세트로 사용하여 검증 평가를 한다. 이 과정을 k번 반복하되 검증 데이터 세트는 매번 다른 것을 사용한다.
  - **KFold 클래스 사용하기**

```
# 1. KFold 객체와 폴드 세트별 정확도를 담을 리스트 객체 생성
kfold = KFold(n_splits=5)
cv_accuracy = []

# 2. split 이용해 n개의 폴드 데이터 세트로 분리
# 폴드 별 학습용, 검증용 테스트의 로우 인덱스를 array로 반환
# 예를 들어, 첫 번째는 0번, 두 변째는 30번, 세 번째는 60번..
for train_index, test_index  in kfold.split(features):

        # 3. 반환된 인덱스를 이용하여 학습용, 검증용 테스트 데이터 ㅔ
    X_train, X_test = features[train_index], features[test
    y_train, y_test = label[train_index], label[test_index

    # 4. 학습 및 예측
    dt_clf.fit(X_train , y_train)
    pred = dt_clf.predict(X_test)
    n_iter += 1

    # 5. 반복 시마다 정확도 측정
    accuracy = np.round(accuracy_score(y_test,pred), 4)
    cv_accuracy.append(accuracy)
```

  - **StratifiedKFold 클래스 사용하기**
    - 불균형한 분포도를 가진 label 데이터 집합을 위한 K 폴드 방식
    - 원본 데이터의 label 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습과 검증 데이터 세트를 분배한다.
    - 학습 레이블 데이터의 분포가 “사과:딸기:포도=1:1:1” 이었다면 검증 레이블 데이터 값도 동일한 비율로 할당한다.

🔪 간편한 교차 검증 → cross_val_score

```
cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')
```

파라미터	의미
estimator	분류 알고리즘 클래스인 Classifier 또는 회귀 알고리즘 클래스인 Regressor
X	feature 데이터 세트
y	label 데이터 세트
scoring	예측 성능 평가 지표
cv	교차 검증 폴드 수

2.3.4 하이퍼 파라미터 튜닝 → GridSearchCV

- 하이퍼 파라미터를 순차적으로 입력하면서 교차 검증을 기반으로 최적의 파라미터를 찾게 해준다.

파라미터	의미
estimator	classifier, regressor, pipeline이 사용될 수 있다.
param_grid	key+리스트 값이르 가지는 딕셔너리가 주어진다.
scoring	예측 성능을 측정할 평가 방법 지정
cv	교차 검증으 위해 분할되는 학습/테스트 세트의 개수 지정
refit	- 디폴트 True - 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼파라미터로 재학습

1.4 데이터 전처리 (Data Processing)

1.4.1 개요

- 어떤 데이터를 입력하느냐에 따라 결과가 크게 달라질 수 있다.

- 결손값 (NaN, Null) 은 다른 값으로 변환해야 한다.
- 문자열 입력 값을 숫자형으로 변환해야 한다.

1.4.2 데이터 인코딩

🥿레이블 인코딩

- 문자열을 숫자값으로 변환
- `LabelEncoder` 를 객체로 생성 수 `fit` 과 `transform` 호출한다.
- e.g. 냉장고 1, 컴퓨터 2, 선풍기 3

🥿원 핫 인코딩

- feature 값의 유형에 따라 새로운 feature를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0 표시
- 여러 개의 속성 중 단 한 개의 속성만 1로 표시
- `get_dummies` 이용

음식		음식.치킨	음식.피자	음식.떡볶이
치킨		1	0	0
피자	➡	0	1	0
떡볶이		0	0	1
피자		0	1	0

```
import pandas as pd
df=pd.DataFrame({'item':['치킨', '피자', '떡볶이']})
pd.get_dummies(df)
```

1.4.3 피처 스케일링과 정규화

🔪피처 스케일링

- 서로 다른 변수의 값 범위를 일정한 수주능로 맏추는 작업
- 1) 표준화
  - 데이터의 feature 각각을 평균이 0이고 분산이 1인 정규 분포로 변환
  - `StandardScaler` 객체 생성 후 `fit` 과 `transform` 메서드에 변환 대상 피처 데이터 세트 입력

```
from sklearn.preprocessing import StandardScaler

# StandardScaler 객체 생성
scaler=StandardScaler()

# StandardScaler 로 데이터 세트 변환. fit 과 transform 호출
scaler.fit(iris_df)
iris_scaled=scaler.transform(iris_df)
```

- 2) 정규화
  - 서로 다른 feature 의 크기를 통일하기 위해 크기를 변환
  - 개별 데이터의 크기를 모두 똑같은 단위로 변경
  - `MinMaxScaler` 객체 생성 후 `fit` 과 `transform` 메서드에 변환 대상 피처 데이터 세트 입력

```
from sklearn.preprocessing import MinMaxScaler

# StandardScaler 객체 생성
scaler=MinMaxScaler()

# StandardScaler 로 데이터 세트 변환. fit 과 transform 호출
scaler.fit(iris_df)
iris_scaled=scaler.transform(iris_df)
```

- 유의점
  - 학습 데이터 세트로 fit 과 transform 적용하면 테스트 데이터 세트에 별도의 fit 을 수행하지 않고, 그 결과 그대로 테스트 데이터 세트에 적용해야 한다.
  - 학습과 테스트 데이터의 스케일링 기준 정보가 서로 같아야 한다.
  - 학습과 테스트 데이터 분리하기 전에 전체 데이터에 스케일링을 적용하자.

3️⃣ 평가

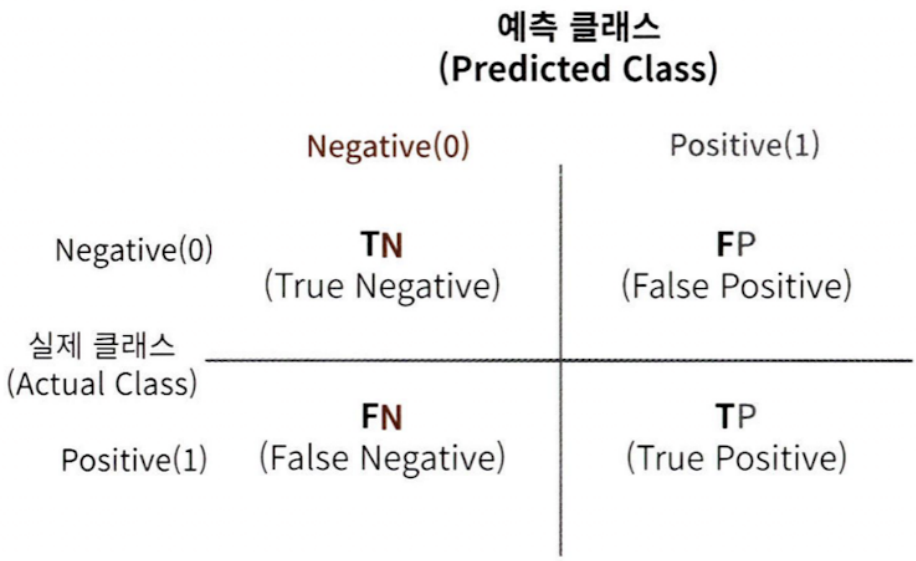
3.1 정확도

3.1.1 개요

$$\text{정확도(Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

- 이진분류의 경우 데이터의 구성에 따라 ML 모델의 성능이 왜곡될 수 있기 때문에 정확도 수치 하나만 가지고 성능을 평가하지 않는다.

3.2 오차 행렬



- e.g. FN : 예측값을 negative 로 예특했는데 실제 값은 positive
- 이진 분류의 예측 오류가 얼마인지와 더물어 어떠한 유형의 예측 오류가 발생하고 있는지를 나타내는 지표
- `confusion_matrix(실제 결과, 예측 결과)`
- 정확도 = 예측 결과와 실제 값이 동일한 건수 / 전체 데이터 수

$$\frac{(TN+TP)}{(TN+FP+FN+TP)}$$

3.3 정밀도와 재현율

3.3.1 정밀도

$$\frac{TP}{(FP+TP)}$$

- 예측을 positive 로 한 대상 중에 실제값이 positive인 비율
- 정밀도가 중요 지표인 예시 → 일반 메일을 스팸 메일로 분류
- FP을 낮추는 데 초점

3.3.2 재현율

$$\frac{TP}{(FN+TP)}$$

- 실제 값이 positive 인 대상 중에 예측값이 positive인 비율
- 민감도 (sensitiviy) 또는 TPR (True Positive Rate) 이라고도 불린다.
- 재현율이 중요 지표인 예시 → 실제 암 환자를 음성으로 잘못 판단
- FN을 낮추는 데 초점

3.3.3 정밀도 / 재현율 트레이드오프

정밀도와 재현율은 상호 보완적인 지표이기 때문에 한 쪽을 높이면 다른 하나의 수치는 떨어진다.

🥿 개별 데이터별 예측 확률 반환 메서드

- `predict_proba(테스트 feature 데이터 세트)`
  - 사이킷런 분류 알고르짐은 예측 데이터가 특정 레이블에 속하는지를 계산하기 위해 먼저 개별 레이블별로 결정 확률을 구한다. 그리고 그 중에 예측 확률이 큰 값으로 예측한다.
  - e.g. 이진 분류 모델에서 특정 데이터가 0이 될 확률이 10%, 1이 될 확률이 90% 로 예측되었다면 최종적으로 1로 예측한다.
  - 학습이 완료된 Classifier 객체에서 호출 가능

🥿 **트레이드오프 방식**

1. threshold 변수를 특정 값으로 설정
2. `Binarizer` 클래스를 객체로 생성
3. `fit_transform` 메서드 이용해 ndarray 입력
4. ndarray의 값이 지정된 threshold 보다 같거나 작으면 0값으로, 크면 1값으로 변환해 반환

🥿 임곗값 변화에 따른 평가 지표

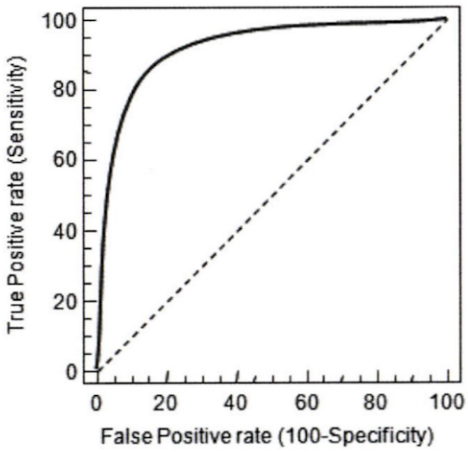
`precision_recall_curve` : 정밀도와 재현율의 임계값에 따른 값 변화를 곡선 형태의 크래프로 시각화

3.4 F1 스코어

- 정밀도와 재현율을 결합한 지표이다.
- 어느 한 쪽으로 치우치지 않는 수치를 나타낼 때 높은 값이 나온다.
- `f1_score`

3.5 ROC 곡선과 AUC

3.5.1 ROC 곡선



〈 ROC 곡선 예시 〉

- Receiver Operation Characteristic Curve
- FPR (False Positive Rate) 이 변할 때 TPR (True Positive Rate, 재현율 / 민감도) 이 어떻게 변하는지를 나타내는 곡선
- TNR (True Negative Rate, 특이성) 은 민감도에 대응하는 지표

$$FPR = \frac{FP}{(FP + TN)} = 1 - TNR$$

- 가운데 직선에 멀어질수록 성능이 뛰어난 것이다.
- FPR 을 0부터 1까지 변경 (=분류 결정 임계값을 변경) 하면서 TPR의 변화 값을 구한다.
  - FPR 0으로 만들기 → 임계값을 1로 지정
  - FPR 1로 만들기 → 임계값을 0으로 지정 (TN을 0으로)
- `roc_curve`

3.5.2 AUC

- Area Under Curve
- ROC 곡선 밑의 면적을 구한 것
- 1에 가까울수록 좋은 수치