2장 사이킷런으로 시작하는 머신러닝

Ch	2
날짜	@2025년 9월 15일
카테고리	개념 정리

01 사이킷런 소개와 특징

scikit-learn

- 가장 파이썬스러운 API

- 머신러닝을 위한 다양한 알고리즘 / 개발 위한 프레임워크 & API 제공
- 매우 많은 환경에서 사용 = 성숙한 라이브러리
- Anaconda 설치시, 기본으로 사이킷런까지 설치 완

```
import sklearn

print(sklearn.__version__)
```

---

02 첫 번째 머신러닝 만들어보기 - 붓꽃 품종 예측

붓꽃 데이터 세트

: 꽃잎의 길이,너비 / 꽃받침의 길이, 너비 feature 기반으로 꽃의 품종 예측

분류 (Classification)

- 대표적인 지도학습 (Supervised Learning) 방법 중 하나
  - 지도학습: 학습을 위한 feature와 분류 결정값인 label data로 모델 학습 ⇒ 미지의 label 예측
  - 명확한 정답이 주어진 데이터 먼저 학습 → 미지의 정답 예측
  - 학습 데이터 세트 (for 학습) / 테스트 데이터 세트 (for 머신러닝 모델의 예측 성능 평가)

모듈 임포트

- `sklearn`
- `sklearn.datasets` 자체 제공된 데이터 세트 생성 모듈의 모임
- `sklearn.tree` 트리 기반 ML 알고리즘 구현한 클래스의 모임
- `sklearn.model_selection` 학습 데이터와 검증 데이터, 예측 데이터 - 데이터 분리 / 최적의 하이퍼 파라미터로 평가하기 위한 다양한 모듈의 모임
  - 하이퍼 파라미터: 최적의 학습을 위해 직접 입력하는 파라미터 ⇒ 성능 튜닝

실습

- `load_iris()` 데이터 세트 생성

- `DecisionTreeClassifier` ML 알고리즘 의사 결정 트리

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

- `train_test_split()` 학습 데이터와 테스트 데이터 분리하는 함
  - feature data set, label data set, ratio, random state number

1. 데이터 세트 분리 (학습 / 테스트)
2. 모델 학습
3. 예측 수행
4. 평가 (예측 결과값 pred VS 실제 결과값 label)

```
import pandas as pd

# 붓꽃 데이터 세트 로딩
iris = load_iris()

# iris.data는 Iris 데이터 세트에 feature만으로 된 데이터를 numpy로 가짐
iris_data = iris.data

# iris.target은 붓꽃 데이터 세트에서 label 데이터를 numpy로 가짐
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)
```

```
# 데이터 분할

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)


# 학습 완료된 DecisionTreeClassifier 객체
# test data set로 예측 수행
pred = dt_clf.predict(X_test)
```

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

---

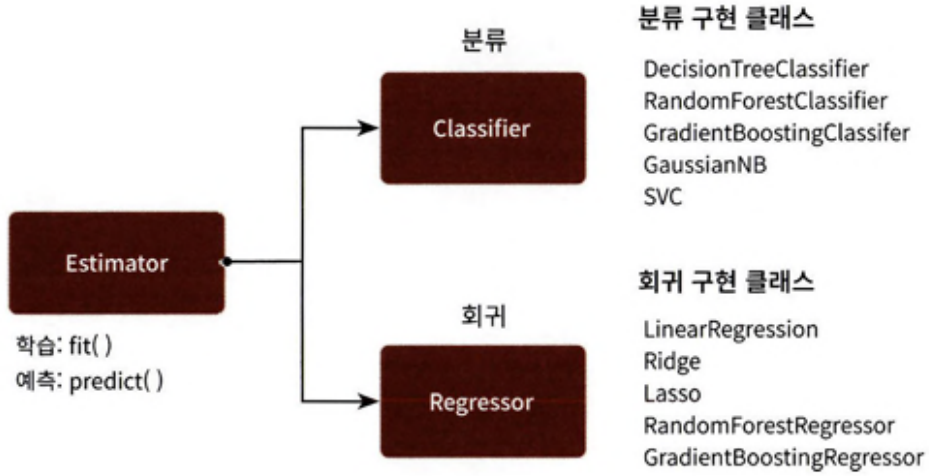
03 사이킷런의 기반 프레임워크

Estimator / fit() / predict() 메서드

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

⇒ 지도학습 - 분류 Classification / 회귀 Regression

- **Estimator**
  - `Classifier` : 분류 알고리즘을 구현한 클래스
  - `Regresor` : 회귀 알고리즘을 구현한 클래스



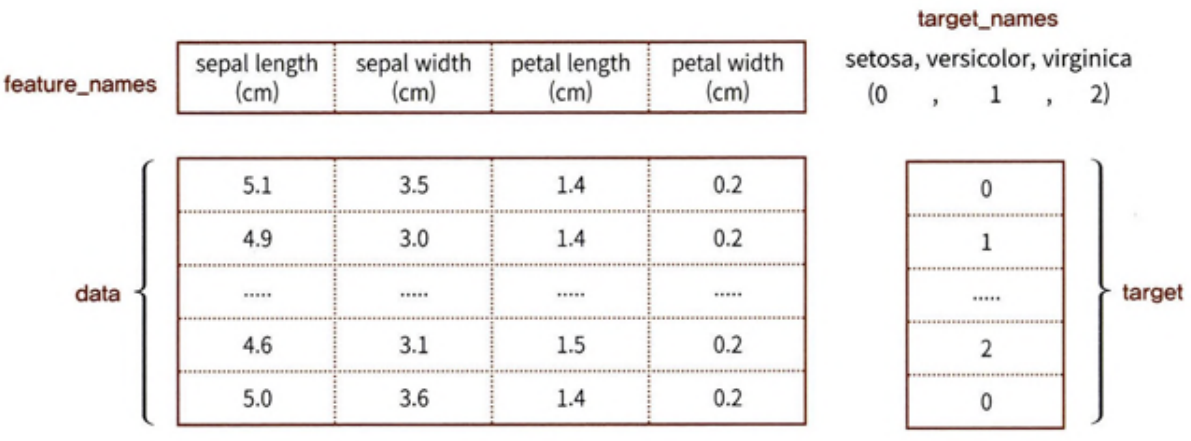
사이킷런 주요 모듈

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 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위대로 선택하는 기능 제공
	sklearn.feature_extraction	텍스트/이미지 데이터의 벡터화된 피처 추출 (예: Count Vectorizer, TF-IDF, 이미지 벡터화)
피처 처리 & 차원 축소	sklearn.decomposition	PCA, NMF, Truncated SVD 등을 통한 차원 축소 기능 제공
데이터 분리, 검증 & 파라미터 튜닝	sklearn.model_selection	교차 검증, 학습/테스트 데이터 분리, Grid Search로 최적 파라미터 탐색 기능 제공
평가	sklearn.metrics	분류/회귀/클러스터링 등 다양한 성능 평가 지표 제공 (Accuracy, Precision, Recall, ROC-AUC, RMSE 등)
ML 알고리즘	sklearn.ensemble	앙상블 알고리즘 제공 (랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등)

분류	모듈명	설명
	`sklearn.linear_model`	선형 회귀, 릿지(Ridge), 라쏘(Lasso), 로지스틱 회귀, SGD 등 제공
	`sklearn.naive_bayes`	나이브 베이즈 알고리즘 제공 (가우시안 NB, 다항 분포 NB 등)
	`sklearn.neighbors`	최근접 이웃 알고리즘 (K-NN 등) 제공
	`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 변환과 ML 알고리즘 학습/예측 과정을 함께 묶어서 실행 가능

내장된 예제데이터 세트

- 분류와 클러스터링을 위한 표본 데이터 생성기
  - `datasets.make_slassifications()` 분류 위한 데이터 무작위 생성
  - `datasets.make_blobs()` 클러스터링 위한 데이터 무작위 생성
- 데이터 구성
  - `data` 데이터 세트 (ndarray)
  - `target` 분류 시 레이블 값 / 회귀에서 숫자 결과값 (ndarray, list)
  - `target_names` 개별 레이블의 이름 (ndarray, list)
  - DESCR 데이터 세트 설명 (string)



04 Model Selection 모듈

- 학습 데이터와 테스트 데이터 세트 분리
- 교차 검증 분할 및 평가
- Estimator의 하이퍼 파라미터 튜닝을 위한 함수, 클래스 제공

학습/테스트 데이터 세트 분리 `train_test_split()`

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

dt_clf = DecisionTreeClassifier( )
iris_data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.targe
t, test_size=0.3, random_state=121)


dt_clf.fit(X_train, y_train)
pred = dt_clf.predict(X_test)
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))
```

교차검증

- 과적합 취약 → 별도의 테스트용 데이터 필요
- 데이터 편중을 막기 위해 별도의 여러 세트로 구성된 학습 데이터 세트 & 검증 데이터 세트에서 학습과 평가 수행
- → 각 세트에서 수행한 평과 결과에 따라 하이퍼 파라미터 튜닝 등 모델 최적화 용이

K-fold 교차 검증

- K개의 데이터 폴드 세트 만들어서 K번 만큼 학습과 검증 평가 반복
- K개의 예측 평가 → 평균

```
n_iter = 0

# KFold 객체의 split()를 호출하면 폴드별 학습용, 검증용 테스트의 로우 인덱스를 arr
ay로 반환
for train_index, test_index in kfold.split(features):
# kfold.split( )으로 반환된 인덱스를 이용해 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]

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

    # 반복 시마다 정확도 측정
    accuracy = np.round(accuracy_score(y_test, pred), 4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]
    print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기:
{3}'.format(n_iter, accuracy, train_size, test_size))
    print('#{0} 검증 세트 인덱스:{1}'.format(n_iter, test_index))

cv_accuracy.append(accuracy)
```

```
# 개별 iteration별 정확도를 합하여 평균 정확도 계산
print('\n## 평균 검증 정확도:', np.mean(cv_accuracy))
```

1. 폴드 세트 설정
2. for 루프에서 반복으로 학습 / 테스트 데이터의 인덱스 추출
3. 반복적으로 학습 & 예측 수행 → 예측 성능 반환

Stratified K fold

- 불균형한(imbalanced) 분포도를 가진 레이블 데이터 집합 위한 K fold 방식
- 특정 레이블 값이 많거나 매우 적어서 분포가 한쪽으로 치우는 경우
  - ex) 대출 사기 데이터 (대부분이 정상 대출, 비정상 대출이 중요 피처 값)
  - 원본 데이터와 유사한 대출 사기 레이블 값의 분포를 학습/테스트 세트에서 유지 필요
- 원본 데이터의 레이블 분포 먼저 고려 → 이 분포와 동일하게 학습/검증 데이터 세트 분

cross_val_score() - 교차 검증 간편하게

▼ k fold 방식을 한꺼번에 수행해주는 API

1. 폴드 세트 설정
2. for 루프에서 반복으로 학습 / 테스트 데이터의 인덱스 추출
3. 반복적으로 학습 & 예측 수행 → 예측 성능 반환

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

GridSearchCV - 교차 검증과 최적 하이퍼 파라미터 튜닝을 한 번

- 교차 검증을 기반으로 하이퍼 파라미터의 최적 값 찾게 함
- 사용자가 튜닝하고자 하는 여러 종류의 하이퍼 파라미터 다양하게 테스트

- 주요 파라미터
  - estimator： classifier, regressor, pipeline이 사용될 수 있습니다.

    - param_grid： key + 리스트 값을 가지는 딕셔너리가 주어집니다. estimator의 튜닝을 위해 파라미터명과 사용될 여러 파라미터 값을 지정합니다.
    - scoring： 예측 성능을 측정할 평가 방법을 지정합니다. 보통은 사이킷런의 성능 평가 지표를 지정하는 문자열（예: 정확도의 경우 'accuracy'）로 지정하나 별도의 성능 평가 지표 함수도 지정할 수 있습니다.
    - cv： 교차 검증을 위해 분할되는 학습/테스트 세트의 개수를 지정합니다.
    - refit： 디폴트가 True이며 True로 생성 시 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습시킵니다
- 주요 칼럼 의미
    - params: 적용된 개별 하이퍼 파라미터값
    - rank_test_score: 하이퍼 파라미터 별로 성능이 좋은 score 순위
    - mean_test_score: 개별 하이퍼 파라미터별로 CV의 폴딩 테스트 세트에 대해 총 수행한 평가 평균값

---

5 데이터 전처리

- NaN, Null 값 허용x → 고정된 다른 값으로 변환
- 사이킷런에서) 문자열 값을 입력값으로 허용X → 인코딩 ⇒ 숫자형 변환 (feature vectorization)

데이터 인코딩

레이블 인코딩

- 카테고리 피처 → 코드형 숫자값으로 변환
- 기존 문자열값 확인: LabelEncoder 객체의 `classes` 속성값

- 레이블 인코딩의 문제: 숫자 값의 대소 → 가중치 부여되는 문제
- 선형회귀 ML 에는 적용 X / 트리 계열 ML O

원-핫 인코딩

- 피처 값의 유형에 따라 새로운 피처 추가 → 고유 값에 해당하는 칼럼에 1, 나머지 0 표시
- `OneHotEncoder` 클래스
- 주의
  - 입력값으로 2차원 데이터 필요
  - 변환 값 (희소 행렬) ⇒ toarray()로 밀집 행렬로 변환
- `get_dummies`

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

items=['TV', '냉장고', '전자레인지','컴퓨터', '선풍기', '선풍기','믹서', '믹서']


# 2차원 ndarray로 변환합니다.
items = np.array(items).reshape(-1, 1)
# 원-핫 인코딩을 적용합니다.
oh_encoder = OneHotEncoder()
oh_encoder.fit(items)
oh_labels = oh_encoder.transform(items)
# OneHotEncoder로 변환한 결과는 희소행렬이므로 toarrayO를 이용해 밀집 행렬로 변환.
print('원-핫 인코딩 데이터')
print(oh_labels.toarray())
print('원-핫 인코딩 데이터 차원')
print(oh_labels.shape)
```

피처 스케일링 / 정규화

- 표준화
  : 데이터의 피처 각각이 평균이 0, 분산이 1인 가우시안 정규 분포 가진 값으로 변환
- 정규화
  : 서로 다른 피처의 크기를 통일하기 위해 크기 변환

StandardScaler - 표준화

```
from sklearn.preprocessing import StandardScaler

# StandardScaler객체 생성
scaler = StandardScaler()

# StandardScaler로 데이터 세트 변환. fit( )과 transform( ) 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

# transform( ) 시 스케일 변환된 데이터 세트가 NumPy ndarray로 반환돼 이를 DataFrame으로 변환
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print ('feature 들의 평균 값')
print(iris_df_scaled.mean())
print('\nfeature 들의 분산 값')
print(iris_df_scaled.var())
```

MinMaxScaler

- 데이터값을 0~1 사이 범위값으로 변환 (음수인 경우 -1~1)

```
from sklearn.preprocessing import MinMaxScaler

# MinMaxScaler객체 생성
scaler = MinMaxScaler()

# MinMaxScaler로 데이터 세트 변환. fit()과 transform() 호출
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

# transform() 시 스케일 변환된 데이터 세트가 NumPy ndarray로 반환돼 이를 DataFrame으로 변환
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature 들의 최솟값')
print(iris_df_scaled.min())
```

```
print('\nfeature들의 최댓값')
print(iris_df_scaled.max())
```

학습 데이터와 테스트 데이터의 스케일링 변환 시 유의점

- `fit_transform()` fit(), transform() 한꺼번에
- 테스트 데이터 세트로는 다시 fit() XXX
  >> 다시 새로운 스케일링 시준 정보를 만들면, 학습데이터 & 테스트데이터 기준 정보가 달라짐
  >> 올바른 예측 결과 X

스케일링 변환 유의점

1. 전체 데이터 스케일링 변환 후 >> train_test_split
2. OR 테스트 데이터 변환시에는 이미 fit()된 Scaler 객체 이용하여 transform()으로 환

---

6 사이킷런으로 수행하는 타이타닉 생존자 예측

■ Passengerid： 탑승자 데이터 일련번호
■ survived： 생존 여부, 0 = 사망, 1 = 생존
■ pdass： 티켓의선실 등급, 1 = 일등석,2 = 이등석,3 = 삼등석
■ sex： 탑승자 성별
■ name： 탑승자 이름
■ Age： 탑승자 나이
■ sibsp： 같이 탑승한 형제자매 또는 배우자 인원수
■ parch: 같이 탑승한 부모님 또는 어린이 인원수
■ ticket: 티켓 번호
■ fare： 요금
■ cabin： 선실번호
■ embarked： 중간 정착 항구 C = Cherbourg, Q = Queenstown, S = Southampton