02

Scikit-learn package

Scikit-learn

Scikit-learn 의 경우 machine learning 을 위한 매우 다양한 알고리즘과 개발을 위해 편리한 프레임워크와 API 를 제공하고 있기 때문에 python machine learning 라이브러리 중 가장 많이 사용됩니다.

Anaconda 환경 상에서는 기본적으로 scikit-learn 까지 설치가 완료되어 있는 상태이기 때문에 별도의 설치가 필요 없습니다. 하지만 별도 설치가 필요한 경우 아래와 같은 명령어로 설치 가능합니다.

```
conda install scikit-learn
```

pip 로 설치하면 다음과 같이 설치 가능합니다.

```
pip install scikit-learn
```

Machine learning 은 1) 데이터 전처리, 2) 학습/평가용 데이터 분리를 거쳐 3) 모델 학습을 진행하고, 4) 평가용 데이터에 대한 예측을 수행합니다.

1) 데이터 전처리
   - 오류 데이터 보정 및 결손 값 처리
   - 레이블 encoding (One hot coding 등)
   - 데이터 정규화(Normalization)/스케일링(Scaling)
2) 학습/평가용 데이터 분리
   - k-fold 세트 분리 교차검증(cross validation)

3) 모델 학습

- 지도/비지도 학습

4) 예측

붓꽃 분류 예제로 미리 보는 scikit-learn

첫 번째로 만들어볼 machine learning 모델은 분류(Classification) 모델입니다. 붓꽃의 꽃잎 높이, 너비 등의 길이 데이터로 품종을 분류하는 실습입니다.

분류는 대표적인 지도학습 방법 중 하나입니다. 지도학습은 명확한 정답이 주어진 데이터를 먼저 학습한 뒤 미지의 정답을 예측하는 방식입니다. 이때 학습을 위해 주어진 데이터를 학습 데이터, 모델의 예측 성능을 평가하기 위해 별도로 주어진 데이터를 평가 데이터로 지칭합니다. 즉 지도학습은 학습을 위한 다양한 피처와 분류 결정 값인 레이블 (label) 데이터로 모델을 학습한 뒤, 별도의 평가 데이터에서 미지의 레이블을 예측하는 방식입니다.

- 필요 scikit-learn module
  - sklearn.datasets: scikit-learn 자체제공 데이터, 붓꽃데이터셋 생성(load_iris)
  - sklearn.tree: tree 기반 ML 알고리즘, 의사결정트리(DecisionTreeClassifier)
  - sklearn.model_selection: 학습/평가 데이터 분리, train_test_split
  - sklearn.metrics: 정확도 측정, accuracy_score

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

- 붓꽃 데이터 가져오기

```
iris = load_iris()
iris_features = iris.feature_names   # iris 데이터셋의 feature 이름 목록
iris_data = iris.data   # iris 데이터셋에서 각 feature 데이터
iris_label = iris.target   #iris 의 label 값
```

- 붓꽃 데이터 시각화

  데이터 프레임을 활용한 붓꽃 데이터 시각화 결과는 아래와 같습니다. 붓꽃 데이터의 feature 는 sepal length/width, petal width/length 으로 구성되어 있고, 데이터프레임에 label 행을 추가한 모습입니다.

```
iris_df = pd.DataFrame(data=iris_data, columns=iris_features)
iris_df['label'] = iris.target
iris_df
```

	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
3	4.6	3.1	1.5	0.2	0

- 붓꽃 데이터 학습용/평가용 데이터 분리, train_test_split(params)

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

- ML 알고리즘으로 학습수행

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

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

- 평가용 데이터로 성능 확인

```
pred = dt_clf.predict(X_test)
accuracy_score(y_test, pred)
```

```
0.9333333333333333
```

Scikit-learn 주요 모듈

Machine learning 의 주요 프로세스는 특징 처리(feature processing), ML 알고리즘 학습/예측 수행, 모델 평가의 단계로 구성되어 있습니다.

Scikit-learn 에서는 machine learning 의 프로세스를 지원하기 위해 아래와 같은 주요 모듈들을 제공합니다.

- 주요 모듈

분류	모듈명	설명
예제데이터	sklearn.datasets	예제로 제공하는 데이터
특징 처리	sklearn.preprocessing	데이터 전처리용
	sklearn.feature_selection	영향이 큰 피처를 우선순위대로 선택
	sklearn.feature_extraction	벡터화된 feature 추출
	sklearn.decomposition	PCA 등 차원축소 기능 수행
ML 알고리즘	sklearn.ensemble	앙상블 알고리즘 제공
	sklearn.linear_model	회귀 관련 알고리즘 지원
	sklearn.svm	Support vector machine 알고리즘 제공
	sklearn.tree	의사결정나무 알고리즘 제공
	sklearn.cluster	비지도 군집화 알고리즘 제공
평가	sklearn.metrics	성능 측정을 위한 모듈

- 예제 데이터 로드

▷ (예제데이터).feature_names

```
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

▷ (예제데이터).target_names

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

▷ (예제데이터).data

```
iris.data
```

```
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
```

▷ (예제데이터).target

```
iris.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

- 모델 선택

▷ 학습/평가 데이터 분리 – train_test_split(params)

param 1, 2 : 피처데이터셋, 레이블데이터셋

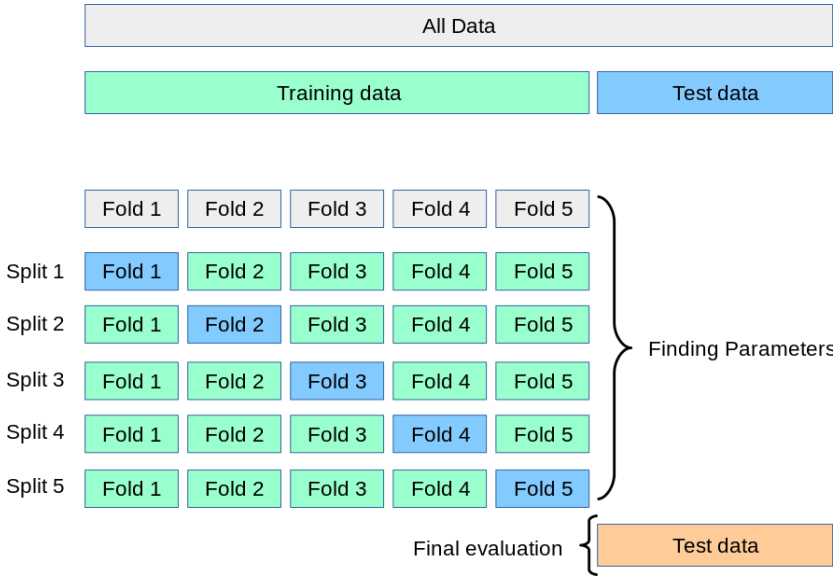
optional params : test_size, shuffle, random_state

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

▷ 교차 검증

교차검증은 overfitting 문제를 막기 위해 학습 데이터를 학습용 및 검증용 데이터로 다시 분할하는 방법입니다.

대표적으로 사용되는 교차검증 기법으로는 k-fold 교차 검증이 있습니다. k 개의 데이터 fold 를 만들어서 k 번 만큼 학습과 검증 평가를 반복적으로 수행하는 방법입니다.




```
from sklearn.model_selection import KFold

kfold = KFold(n_splits = 5)
for train_index, test_index in kfold.split(iris.data):
  # kfold.split() 로 반환된 인덱스를 활용하여 학습/테스트 데이터 추출
  X_train, X_test = iris.data[train_index], iris.data[test_index]
  y_train, y_test = iris.target[train_index], iris.target[test_index]
```

그러나 k-fold 의 경우, 분할된 데이터가 전체 레이블 값의 분포도를 반영하지 못하는 문제를 가지고 있습니다. 이를 해결하여 결국 성능 향상을 이루기 위해서

StratifiedKFold 를 통해 데이터의 전체적인 분포도에 따라 학습/검증 데이터를 나눌 수 있습니다.

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits= 3)
for train_index, test_index in skf.split(iris_df, iris_df['label']):
  label_train = iris_df['label'].iloc[train_index]
  label_test = iris_df['label'].iloc[test_index]
```

```
학습 레이블 분포
 2    34
1    33
0    33
Name: label, dtype: int64
검증 레이블 분포
 1    17
0    17
2    16
```

교차검증을 보다 간편하게 접근할 수 있는 방법도 있습니다. cross_val_score (estimator, X, y, scoring, cv)에서 estimator 는 사이킷런의 ML 알고리즘을, X 는 feature 데이터, y 는 레이블 데이터, scoring 은 예측 성능 평가 지표를, cv 는 교차 검증 fold 수를 의미합니다. cross_val_score()는 지정된 성능 지표 측정값을 반환합니다.

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(dt_clf,iris.data, iris.target, scoring="accuracy", cv=3)
print("교차 검증 별 정확도: ",scores)
```

```
교차 검증 별 정확도:  [0.98 0.92 0.98]
```

- 데이터 전처리
  - 결손 값 처리: 평균값 대체 / feature drop
  - 데이터 encoding

One-hot encoding 은 피처의 고유 값에 해당하는 칼럼에만 1 을 표시하고, 나머지 칼럼에는 0 을 표시하는 방식입니다.

One-hot encoding 은 scikit-learn 에서 OneHotEncoder 클래스로 쉽게 변환이 가능합니다. 기존 라벨들이 숫자 형 값이면서, 2 차원 데이터 형태를 유지하고 있다면 해당 클래스로 one-hot encoding 을 진행할 수 있습니다.

```
from sklearn.preprocessing import OneHotEncoder

items = np.array([1,2,3,4,5])
labels = items.reshape(-1,1)      # 2차원 데이터로 변환
one_hot = OneHotEncoder()
one_hot.fit(labels)
one_hot_labels = one_hot.transform(labels)
print("기존 라벨: ", items)
print("원핫인코딩 결과: ₩n",one_hot_labels.toarray())
```

```
기존 라벨:  [1 2 3 4 5]
원핫인코딩 결과:
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

▷ 특징 스케일링 (feature scaling)

서로 다른 변수 값의 범위를 일정한 수준으로 맞추는 작업을 특징 스케일링이라고 합니다. 대표적인 방법으로는 표준화와 정규화가 있습니다.

표준화(Standardization)

평균 0, 분산 1 인 Gaussian 정규 분포로 변환

$$x_{new} = \frac{x - \mu}{\sigma}$$

정규화(Normalization)

서로 다른 특징의 크기를 통일하기 위해 크기를 변환해주는 개념으로, 모든 값들을 0~1 사이 값으로 변환

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Scikit-learn 선형회귀 등의 모델은 데이터가 Gaussian 분포를 가지고 있다고 가정하고 구현됐기 때문에 사전에 표준화를 적용하는 것은 성능 향상에 중요한 요소가 될 수 있습니다.

사이킷런에서 표준화를 쉽게 지원해주는 클래스는 StandardScaler 입니다.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(iris_df)
iris_df_scaled = pd.DataFrame(data=scaler.transform(iris_df),
                              columns = iris.feature_names)
print("feature 값들의 평균")
print(iris_df_scaled.mean())
print("feature 값들의 분산")
print(iris_df_scaled.var())
```

```
feature 값들의 평균
sepal length (cm)   -1.690315e-15
sepal width (cm)    -1.842970e-15
petal length (cm)   -1.698641e-15
petal width (cm)    -1.409243e-15
dtype: float64
feature 값들의 분산
sepal length (cm)    1.006711
sepal width (cm)     1.006711
petal length (cm)    1.006711
petal width (cm)     1.006711
dtype: float64
```

StandardScaler 를 통해 모든 컬럼 값의 평균이 0 에 아주 가까운 값으로, 분산은 1 에 가까운 값으로 변환되었음을 알 수 있습니다.

다음으로는 MinMaxScaler 입니다. MinMaxScaler 는 데이터 값을 0 과 1 사이의 범위 값으로 변환합니다. 데이터 분포가 Gaussian 분포가 아닐 경우에 적용해볼 수 있습니다.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(iris_df)
iris_df_scaled = pd.DataFrame(data=scaler.transform(iris_df),
                              columns = iris.feature_names)
iris_df_scaled.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

Scikit-learn 실습 – Titanic 생존자 예측

- https://www.kaggle.com/c/titanic/overview