실습1) Linear Regression

June 29, 2022

- **sklearn의 주요 모듈**

모듈명	설명
datasets	분석 실습을 위한 샘플 데이터셋
preprocessing	전처리를 위한 데이터 가공 (코딩변경, 정규화,스케일링등)
model_selection	교차검증을 위한 훈련/평가 데이터 분할. grid search로 파라미터 최적화
metrics	분류, 회귀 등에 대한 다양한 성능지표 제공
ensemble	앙상블 알고리즘 (랜덤포레스트, 에이다부스트, 그래디언트부스트 등)
linear_model	선형회귀관련 알고리즘 (선형회귀, 릿지, 라쏘, 로지스틱 등)
tree	의사결정나무 알고리즘
cluster	클러스터링 알고리즘 (K-평균, 계층형 등)
svm	SVM 알고리즘
naive_bayes	나이브베이즈 알고리즘
decomposition	차원축소 관련 알고리즘(PCA, SVD 등)

- **sklearn의 API 기초**
  - 일반적으로 sklearn estimator API를 이용하는 단계는 다음과 같음.
    1. sklearn으로부터 적절한 머신러닝 모델에 관한 모듈을 import하고 클래스를 선택함.
    2. 선택한 모델 클래스에 원하는 parameter와 hyper parameter 값을 지정하여 인스턴스화함.
    3. 모델 인스턴스의 fit() 메서드를 이용하여 모델을 데이터에 적합시킴.
    4. 적합된 모델을 적용함.
       * **지도학습**의 경우, predict() 메서드를 사용하여 데이터에 대한 레이블을 예측
       * **비지도학습**의 경우, transform() 메서드를 사용하여, 입력 데이터의 차원 변환, 클러스터링, 특성 추출 등의 변환 작업을 적용함.

- **sklearn.linear_model 모듈**: **LinearRegression (선형회귀모델)**
  - 회귀모형의 훈련 : sklearn.linear_model 모듈의 **LinearRegression 클래스**
    * *sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False)*
    * 입력 파라미터
      · fit_intercept : True/False, 절편항 포함 여부
      · normalize : True/False, 입력데이터셋 정규화 여부.
    * 메서드
      · fit(X_train,y_train) 메서드로 모델을 학습.
      · predict(X_test) 메서드로 새로운 입력데이터에 학습된 모델을 적용한 예측.
    * 속성
      · coef_ : fit() 메서드를 수행했을 때 각 feature 변수 별 추정된 가중치(기울기계수)가 배열 형태로 저장.

· intercetp_ : 편향(절편) 추정치

- **sklearn의 datasets 모듈**
  - sklearn.datasets 모듈
    - 예제 데이터셋의 구성
      · datasets.load_boston() : 보스턴의 집 가격 데이터. 회귀용
      · datasets.load_breast_cancer() : 위스콘신 유방암 데이터. 분류용
      · datasets.load_diabetes() :당뇨 데이터. 회귀용
      · datasets.load_digits() :0~9의 숫자 이미지 픽셀 데이터. 분류용
      · datasets.load_iris() :붓꽃 데이터. 분류용
    - 각 예제 데이터는 딕셔너리 형태로 키는 다음과 같이 정의됨.
      · data : 특성 데이터 세트, ndarray
      · target : 분류 데이터의 경우 레이블 값, 회귀 데이터의 경우 결과값, ndarray
      · target_names : 개별 레이블의 이름, ndarray 또는 list
      · feature_names : 특성 변수의 이름, ndarray 또는 list
      · DESCR : 데이터세트 및 각 변수에 대한 설명

- **sklearn의 model_selection 모듈**
  - train_test_split(X, y, . . .) : 훈련/평가 데이터 세트의 분리
    - train_test_split()의 선택 파라미터
      · test_size : 전체 중 평가 데이터의 비중. default는 0.25.
      · random_state : train_test_split()을 호출할 때마다 동일한 훈련/평가용 데이터 세트를 생성하기 위해 주어지는 난수 값.
    - train_test_split()의 반환값
      · 훈련X, 평가X, 훈련y, 평가y의 순서로 데이터 세트가 저장된 튜플을 반환
  - cross_val_score() : K 폴드 교차검증을 간편하게 수행할 수 있는 API를 제공
    - cross_val_score(estimator, X, y=None, scoring=None, cv=None,. . .)의 주요 파라미터
      · estimator : 분류 또는 회귀 알고리즘 클래스 (Classifier 또는 Regressor)
      · X : 특성변수 데이터 세트
      · y : 목표변수 데이터 세트
      · scoring : 예측 성능을 측정할 평가 방법을 지정. 보통은 사이킷런의 성능평가지표를 지정하는 문자열 (예. 'accuracy')로 지정함.
      · cv : 교차 검증 폴드 수
    - cv로 지정된 횟수만큼 scoring 파라미터로 지정된 평가지표로 평가 결과값을 배열로 반환. 일반적으로 이를 평균하여 평균 수치로 활용.

- **sklearn의 metrics 모듈**
  - 머신러닝 알고리즘의 평가지표 계산에 필요한 API를 제공함.
  - 지도학습 중 **회귀** 알고리즘의 경우 다음의 지표가 활용됨.
    - mean_sqaured_error(y_test, y_preds)
    - r2_score(y_test, y_preds)
  - 지도학습 중 **분류** 알고리즘의 평가지표는 추후 논의.

평가방법	sklearn 평가지표 API	scoring 함수 적용 값
MSE	metrics.mean_squared_error	'neg_mean_squared_error'
R squared	metrics.r2_score	'r2'

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_boston
```

- sklearn의 보스턴 주택가격 데이터 : **sklearn.datasets.load_boston()**
  - 타겟 데이터
    - 1978 보스턴 주택 가격(506개 타운의 주택 가격 중앙값 (단위 1,000 달러))
  - 특징 데이터
    - CRIM: 범죄율
    - INDUS: 비소매상업지역 면적 비율
    - NOX: 일산화질소 농도
    - RM: 주택당 방 수
    - LSTAT: 소득이 낮은 사람의 비율
    - B: 인구 중 흑인 비율
    - PTRATIO: 학생/교사 비율
    - ZN: 25,000 평방피트를 초과 거주지역 비율
    - CHAS: 찰스강의 경계에 위치한 경우는 1, 아니면 0
    - AGE: 1940년 이전에 건축된 주택의 비율
    - RAD: 방사형 고속도로까지의 거리
    - DIS: 직업센터의 거리
    - TAX: 재산세율

[2]:
```
boston = load_boston()
boston_DF = pd.DataFrame( boston.data, columns=boston['feature_names'])
boston_DF['PRICE']=boston['target']
boston_DF.head(5)
```

[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

	PTRATIO	B	LSTAT	PRICE
0	15.3	396.90	4.98	24.0
1	17.8	396.90	9.14	21.6
2	17.8	392.83	4.03	34.7
3	18.7	394.63	2.94	33.4
4	18.7	396.90	5.33	36.2

[3]:
```
boston_DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
```

```
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

[4]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

[5]:
```
y_target = boston_DF['PRICE']
X_data = boston_DF.drop(['PRICE'], axis=1, inplace=False)
X_train, X_test, y_train, y_test = train_test_split(
        X_data, y_target, test_size=0.4, random_state=123 )
```

[6]:
```
lr = LinearRegression()
lr.fit ( X_train, y_train )
```

[6]:
```
LinearRegression()
```

[7]:
```
lr.intercept_
```

[7]:
```
25.011405223295796
```

[8]:
```
np.around( lr.coef_, decimals=1 )
```

[8]:
```
array([ -0.1,   0. ,   0.1,   0.4, -13.8,   4.9,  -0. ,  -1.2,   0.3,
        -0. ,  -0.9,   0. ,  -0.5])
```

[9]:
```
coeff = pd.Series( data= np.around( lr.coef_, decimals=2 ), index=X_data.columns␣
 ↪)
coeff.sort_values(ascending=False)
```

[9]:
```
RM          4.94
CHAS        0.45
RAD         0.28
```

```
INDUS       0.09
ZN          0.03
B           0.01
AGE        -0.01
TAX        -0.01
CRIM       -0.10
LSTAT      -0.52
PTRATIO    -0.90
DIS        -1.17
NOX       -13.83
dtype: float64
```

[10]:
```
y_preds = lr.predict( X_test )
mse = mean_squared_error( y_test, y_preds )
rmse = np.sqrt( mse )
rmse
```

[10]: 5.097391401613322

[11]:
```
r2 = r2_score( y_test, y_preds )
r2
```

[11]: 0.69462808741428

[12]:
```
y_train_preds = lr.predict( X_train )
mse_train = mean_squared_error( y_train, y_train_preds )
rmse_train = np.sqrt( mse_train )
rmse_train
```

[12]: 4.534939011357761

[13]:
```
r2_train = r2_score( y_train, y_train_preds )
r2_train
```

[13]: 0.7546781709705461

[14]:
```
from sklearn.model_selection import cross_val_score
neg_mse_scores= cross_val_score(lr, X_data, y_target,
                                scoring='neg_mean_squared_error', cv=5)
rmse_scores = np.sqrt( -1 * neg_mse_scores )
rmse_scores
```

[14]: array([3.52991509, 5.10378498, 5.75101191, 8.9867887 , 5.77179405])

[15]:
```
np.mean( rmse_scores )
```

[15]: 5.828658946215808