

교육 제목	데이터 기반 인공지능 시스템 엔지니어 양성 과정_ 머신러닝
교육 일시	2021년 10월 18일
교육 장소	YGL C-6 학과장 & 자택(디스코드 이용한 온라인)
교육 내용	
오전	<p>지난 시간 Review & 기계학습(Machine Learning ML) 혼공머(혼자 공부하는 머신러닝)</p> <p>03-2. 선형회귀</p> <ul style="list-style-type: none"> • 선형 회귀는 특성과 타깃 사이의 관계를 가장 잘 나타내는 선형 방정식을 찾습니다. 특성이 하나면 직선 방정식이 됩니다. • 선형 회귀가 찾은 특성과 타깃 사이의 관계는 선형 방정식의 계수 또는 기중치에 저장됩니다. 머신러닝에서 종종 가중치는 방정식의 기울기와 절편을 모두 의미하는 경우가 많습니다. • 모델 파라미터는 선형 회귀가 찾은 기중치처럼 머신러닝 모델이 특성에서 학습한 파라미터를 말합니다. • 다항 회귀는 다항식을 사용하여 특성과 타깃 사이의 관계를 나타냅니다. 이 함수는 비선형일 수 있지만 여전히 선형 회귀로 표현할 수 있습니다. <p>scikit-learn</p> <ul style="list-style-type: none"> • LinearRegression은 사이킷런의 선형 회귀 클래스입니다. <p>fit_intercept 매개변수를 True로 지정하면 절편을 학습하지 않습니다. 이 매개변수의 기본 값은 True입니다.</p> <p>학습된 모델의 coef_ 속성은 특성에 대한 계수를 포함한 배열입니다. 즉 이 배열의 크기는 특성의 개수와 같습니다. intercept_ 속성에는 절편이 저장되어 있습니다.</p> <pre># 훈련 세트와 테스트 세트로 나눕니다 train_input, test_input, train_target, test_target = train_test_split(perch_length, perch_weight, random_state=42)</pre> <pre># 훈련 세트와 테스트 세트를 2차원 배열로 바꿉니다 train_input = train_input.reshape(-1, 1) test_input = test_input.reshape(-1, 1)</pre>

```
from sklearn.neighbors import KNeighborsRegressor

knr = KNeighborsRegressor(n_neighbors=3)

# k-최근접 이웃 회귀 모델을 훈련합니다
knr.fit(train_input, train_target)

print(knr.predict([[50]]))

import matplotlib.pyplot as plt

# 50cm 농어의 이웃을 구합니다
distances, indexes = knr.kneighbors([[50]])

# 훈련 세트의 산점도를 그립니다
plt.scatter(train_input, train_target)

# 훈련 세트 중에서 이웃 샘플만 다시 그립니다
plt.scatter(train_input[indexes], train_target[indexes], marker='D')

# 50cm 농어 데이터
plt.scatter(50, 1033, marker='^')
plt.show()

"""\#\# 선형 회귀"""

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

# 선형 회귀 모델을 훈련합니다
lr.fit(train_input, train_target)
```

```
# 50cm 농어에 대해 예측합니다
print(lr.predict([[50]]))

print(lr.coef_, lr.intercept_)

# 훈련 세트의 산점도를 그립니다
plt.scatter(train_input, train_target)

# 15에서 50까지 1차 방정식 그래프를 그립니다
plt.plot([15, 50], [15*lr.coef_+lr.intercept_, 50*lr.coef_+lr.intercept_])

# 50cm 농어 데이터
plt.scatter(50, 1241.8, marker='^')
plt.show()

print(lr.score(train_input, train_target)) # 훈련 세트
print(lr.score(test_input, test_target)) # 테스트 세트
```

```
"""## 다항 회귀"""

train_poly = np.column_stack((train_input ** 2, train_input))
test_poly = np.column_stack((test_input ** 2, test_input))

print(train_poly.shape, test_poly.shape)

lr = LinearRegression()
lr.fit(train_poly, train_target)

print(lr.predict([[50**2, 50]]))

print(lr.coef_, lr.intercept_)
```

지난 시간 Review & 기계학습(Machine Learning ML) 훈공머(혼자 공부하는 머신러닝)

03-3. 특성 공학과 규제

- **다중 회귀**는 여러 개의 특성을 사용하는 회귀 모델입니다. 특성이 많으면 선형 모델은 강력한 성능을 발휘합니다.
- **특성 공학**은 주어진 특성을 조합하여 새로운 특성을 만드는 일련의 작업 과정입니다.
- **릿지**는 규제가 있는 선형 회귀 모델 중 하나이며 선형 모델의 계수를 작게 만들어 과대적합을 완화시킵니다. 릿지는 비교적 효과가 좋아 널리 사용하는 규제 방법입니다.
- **라쏘**는 또 다른 규제가 있는 선형 회귀 모델입니다. 릿지와 달리 계수 값을 아예 0으로 만들 수도 있습니다.
- **하이퍼파라미터**는 머신러닝 알고리즘이 학습하지 않는 파라미터입니다. 이런 파라미터는 사람이 사전에 지정해야 합니다. 대표적으로 릿지와 라쏘의 규제 강도 alpha 파라미터입니다.

오후

pandas

- **read_csv()**는 CSV 파일을 로컬 컴퓨터나 인터넷에서 읽어 판다스 데이터프레임으로 변환하는 함수입니다. 이 함수는 매우 많은 매개변수를 제공합니다. 그중에 자주 사용하는 매개변수는 다음과 같습니다.

sep는 CSV 파일의 구분자를 지정합니다. 기본값은 ‘콤마(.)’입니다.

header에 데이터프레임의 열 이름으로 사용할 CSV 파일의 행 번호를 지정합니다. 기본적으로 첫 번째 행을 열 이름으로 사용합니다.

skiprows는 파일에서 읽기 전에 건너뛸 행의 개수를 지정합니다.

nrows는 파일에서 읽을 행의 개수를 지정합니다.

scikit-learn

- **PolynomialFeatures**는 주어진 특성을 조합하여 새로운 특성을 만듭니다.

degree는 최고 차수를 지정합니다. 기본값은 2입니다.

interaction_only가 True이면 거듭제곱 항은 제외되고 특성 간의 곱셈 항만 추가됩니다. 기본값은 False입니다.

include_bias가 False이면 절편을 위한 특성을 추가하지 않습니다. 기본값은 True입니다.

- **Ridge**는 규제가 있는 회귀 알고리즘인 릿지 회귀 모델을 훈련합니다.

alpha 매개변수로 규제의 강도를 조절합니다. alpha 값이 클수록 규제가 세집니다. 기본값은 1입니다.

solver 매개변수에 최적의 모델을 찾기 위한 방법을 지정할 수 있습니다. 기본값은 'auto'이며 데이터에 따라 자동으로 선택됩니다. 사이킷런 0.17 버전에 추가된 'sag'는 확률적 평균 경사 하강법 알고리즘으로 특성과 샘플 수가 많을 때 성능이 빠르고 좋습니다. 사이킷런 0.19 버전에는 'sag'의 개선 버전인 'saga'가 추가되었습니다.

random_state는 solver가 'sag'나 'saga'일 때 넘파이 난수 시드값을 지정할 수 있습니다.

- **Lasso**는 규제가 있는 회귀 알고리즘인 라쏘 회귀 모델을 훈련합니다. 이 클래스는 최적의 모델을 찾기 위해 좌표축을 따라 최적화를 수행해가는 좌표 하강법 coordinate descent을 사용합니다.

alpha와 random_state 매개변수는 Ridge 클래스와 동일합니다.

max_iter는 알고리즘의 수행 반복 횟수를 지정합니다. 기본값은 1000입니다.

```
"""## 데이터 준비"""
```

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(
    perch_full, perch_weight, random_state=42)
```

```
"""## 사이킷런의 변환기"""
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly = PolynomialFeatures()
poly.fit([[2, 3]])
print(poly.transform([[2, 3]]))

poly = PolynomialFeatures(include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
print(train_poly.shape)

poly.get_feature_names()

test_poly = poly.transform(test_input)
```

```
"""## 다중 회귀 모델 훈련하기"""

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(train_poly, train_target)
print(lr.score(train_poly, train_target))

print(lr.score(test_poly, test_target))

poly = PolynomialFeatures(degree=5, include_bias=False)
poly.fit(train_input)
train_poly = poly.transform(train_input)
test_poly = poly.transform(test_input)
print(train_poly.shape)

lr.fit(train_poly, train_target)
print(lr.score(train_poly, train_target))

print(lr.score(test_poly, test_target))
```

```
"""## 규제"""

from sklearn.preprocessing import StandardScaler
```

```
ss = StandardScaler()
ss.fit(train_poly)
train_scaled = ss.transform(train_poly)
test_scaled = ss.transform(test_poly)

"""## 릿지 회귀"""

from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(train_scaled, train_target)
print(ridge.score(train_scaled, train_target))

print(ridge.score(test_scaled, test_target))
```

```
import matplotlib.pyplot as plt
train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
    # 릿지 모델을 만듭니다
    ridge = Ridge(alpha=alpha)
    # 릿지 모델을 훈련합니다
    ridge.fit(train_scaled, train_target)
    # 훈련 점수와 테스트 점수를 저장합니다
    train_score.append(ridge.score(train_scaled, train_target))
    test_score.append(ridge.score(test_scaled, test_target))

plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.show()

ridge = Ridge(alpha=0.1)
ridge.fit(train_scaled, train_target)
print(ridge.score(train_scaled, train_target))
print(ridge.score(test_scaled, test_target))
```

```
"""## 라쏘 회귀"""

from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(train_scaled, train_target)
print(lasso.score(train_scaled, train_target))

print(lasso.score(test_scaled, test_target))

train_score = []
test_score = []
alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
for alpha in alpha_list:
    # 라쏘 모델을 만듭니다
    lasso = Lasso(alpha=alpha, max_iter=10000)
    # 라쏘 모델을 훈련합니다
    lasso.fit(train_scaled, train_target)
    # 훈련 점수와 테스트 점수를 저장합니다
    train_score.append(lasso.score(train_scaled, train_target))
    test_score.append(lasso.score(test_scaled, test_target))

plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.show()

lasso = Lasso(alpha=10)
lasso.fit(train_scaled, train_target)
print(lasso.score(train_scaled, train_target))
print(lasso.score(test_scaled, test_target))

print(np.sum(lasso.coef_ == 0))
```