Chapter 03. 회귀 알고리즘과 모델 규제 . 농어의 무게를 예측하라!

03-1. k-최근접 이웃 회귀.

무언가 고르는 문제가 아니고 무게를 예측해야 한다. → 무게는 어떤 숫자도 가능.

↳ "회귀"

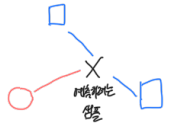
↳ 무게를 정확히 측정한 샘플 필요. → 농어 샘플 5마리 get!

지도학습 알고리즘

- 분류 : 샘플을 몇 개의 클래스 중 하나로 분류하는 문제. → 2장에서 다룸.
- 회귀 : 임의의 어떤 수를 예측하는 문제. → 두 변수 사이의 상관관계를 분석하는 방법.
  ↳ 농어의 무게 예측.

k-최근접 이웃 분류 알고리즘

1. 예측하려는 샘플에 가장 가까운 샘플 k개를 선택.
2. 샘플들의 클래스를 확인하여 다수 클래스를 새로운 샘플의 클래스로 예측 한다.



□클래스 2개 → 다수 클래스.

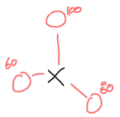
○클래스 1개

∴ X는 □ 클래스로 예측한다.

k-최근접 회귀 알고리즘

↳ 회귀이기 때문에 클래스가 아닌 임의의 수치이다.

1. 예측하려는 샘플에 가장 가까운 샘플 k개 선택.
2. 수치들의 평균을 구한다.



→ 평균 $= \frac{100+80+60}{3}$

$= \frac{240}{3}$

$= 80$ //

〈데이터 준비〉

- 훈련세트 : 농어의 길이만 있어도 무게 예측 가능.

→ 길이 : 특성 , 무게 : 타깃.

- 산점도

하나의 특성 사용. ↳ 특성 데이터 (길이) : x축.
↳ 타깃 데이터 (무게) : y축.

→ 맷플롯립 import 후 scatter() 함수 사용해서 산점도 그림.

사이킷런의 train_test_split() 함수 : 훈련 세트 , 테스트 세트 나눔.

2차원 배열 사용해야 함.

∴ 1차원 배열 → 2차원 배열. , [1, 2, 3] → [[1], [2], [3]]

크기 : (3,) 크기 : (3, 1)

2차원 배열로 만들기 위해 여러로 리스트 묶을 수 있다.
↳ 배열을 나타내는 방법만 바뀜 , 원소 그대로.

특성을 하나만 사용 ⇒ 샘플은 2차원 배열로 만들어야 함. → 넘파이 배열은 크기를 바꿀 수 있는 reshape() 메서드 제공.

※ reshape() 메서드

: 배열의 크기 바꿈.

↳ 바꾸려는 배열의 크기를 지정할 수 있다.

```
ex). test_array = np.array([1,2,3,4])      // (4, ) 배열.
     test_array = test_array.reshape(2,2)  // (2,2) 크기로 바꿈.
     print(test_array)                     // (2,2) 출력.
```

※ 넘파이는 배열의 크기를 자동으로 지정하는 기능도 제공한다.

크기 -1 : 나머지 원소 개수를 모두 지정하는 의미.

```
ex). train_input = train_input.reshape(-1,1)
     test_input = test_input.reshape(-1,1)      // (42, 1) , (14, 1)
     print(train_input, test_input)
```

reshape(-1,1) → 배열의 전체 원소 개수 일일이 계산할 필요 X.

```
from sklearn.neighboors import KNeighborsRegressor  k-최근접 이웃 회귀 알고리즘을 구현할 클래스 KNeighborsRegressor.

knr = KNeighborsRegressor()  객체 생성

knr.fit(train_input, test_target)  k-최근접 이웃 회귀 모델 훈련.

print(knr.score(test_input, test_target))
```

⇒ 0.992809406101064

- 분류 : 정확도 (정답을 맞힌 개수 비율)
- 회귀 : 결정계수 (R^2)

* $R^2 = 1 - \frac{(\text{타깃}-\text{예측})^2\text{의 합}}{(\text{타깃}-\text{평균})^2\text{의 합}}$

↳ 타깃의 평균 정도를 예측하는 수준 (분자, 분모가 비슷해짐) ⟶ R^2는 0에 가까워짐.

↳ 예측이 타깃에 아주 가까워지면 (분자가 0에 가까워짐) ⟶ R^2는 1에 가까워짐.

* mean_absolute_error : 타깃과 예측의 절댓값 오차를 평균하여 반환함.

훈련세트와 테스트세트의 점수를 비교할 때

과대적합 훈련세트가 너무 높으면 "과대적합"

↳ 훈련세트에서 점수가 좋았는데 테스트 세트에서는 점수가 괜찮지 않다. ⇒ 훈련세트에 과대적합이다.

⇒ 훈련 세트에만 잘 맞는 모델. ⇝ 원치 X.

과소적합 둘 중 하나나 수두 모두 낮으면 "과소적합"

↳ 훈련세트보다 테스트 세트의 점수가 높거나 두 점수가 모두 너무 낮다. ⇒ 훈련세트에 과소적합이다.

⇒ 모델이 너무 단순하여 훈련 세트에 적절히 훈련되지 않은 경우.

⇓

과소적합 해결방법 : 모델을 더 복잡하게 생성.

↓

k-최근접 이웃 알고리즘 모델에서 더 복잡하게 만드는 방법 : 이웃의 개수 k를 줄임.

* 이웃의 개수↓ : 훈련 세트에 있는 국지적인 패턴에 민감해짐.

이웃의 개수↑ : 데이터 전반에 있는 일반적인 패턴을 따름.

03-2. 선형회귀.

선형회귀 - 널리 사용되는 대표적인 회귀 알고리즘.

: 특성이 하나인 경우 어떤 직선을 훈련하는 알고리즘.
↳ 특성을 가장 잘 나타낼 수 있는 직선.

※ 사이킷런은 sklearn.linear_model 패키지 아래 LinearRegression 클래스로 선형 회귀 알고리즘 구현.

↳ 가장 간단한 직선 방정식 사용.
↳ LinearRegression 클래스가 이 데이터에 가장 잘 맞는 a 와 b를 찾는다.

선형 회귀 모델의 coef_ 와 intercept_ 속성에 저장됨. → coef_ 와 intercept_ 는 모델 파라미터이다.
- coef_ : 특성에 대한 계수를 포함한 배열 (크기 : 특성의 개수)
- intercept_ : 절편이 저장됨.

선형 회귀가 찾은 것과 같이 머신러닝 모델이 특성에서 학습한 파라미터.

다항회귀 : 다항식을 활용한 선형 회귀

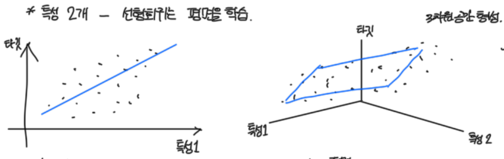
산점도 — 곡선 ⇒ 직선을 곡선으로 변경. (2차방정식 그래프가 갖을 제곱항이 훈련세트에 추가되어야 한다.)

2차방정식 → 선형회귀 진행.

03-3. 특성 공학과 규제

다중회귀 : 여러 개의 특성을 사용한 선형회귀

* 특성 2개 — 선형회귀는 평면을 학습.



↳ 직선 ↳ 평면.

⤳ 회귀 방정식

'타깃 = a × 특성1 + b × 특성2 + 절편

* 특성 3개 — X

↳ 3차원 공간 이상을 그리거나 상상 X

특성공학 : 기존의 특성을 사용해 새로운 특성을 뽑아내는 작업.

* 판다스 : 데이터 분석 라이브러리.

* 데이터 프레임. : 판다스의 핵심 데이터 구조.

    ↳ 넘파이 배열과 비슷하게 다차원 배열을 다룰 수 있지만 훨씬 더 많은 기능 제공.

판다스에서 파일 읽는 법

: read_csv() 함수에 주소 넣음 ⇒ pd.read_csv(주소) 데이터 프레임 만듦.

to_numpy() 매서드 사용해서 넘파이 배열로 바꿈. ex). perch_full = df.to_numpy()

csv 파일 — 판다스 데이터 프레임 — 넘파이 배열.

csv파일 ⟶ pd.read_csv() ⟶ to_numpy()

변환기 : 특성을 만들거나 전처리하기 위해 제공하는 다양한 클래스

⤳ 사이킷런에는 변환기라 부르는 특성을 만들거나 전처리하는 클래스가 있다.

↳ 변환기 클래스는 모두 fit() (훈련), transform() (변환) 메서드를 제공. ⟶ 하나로 붙인 fit_transform 메서드 존재.

PolyomialFeatures 클래스.

↳ sklearn.preprocessing 패키지에 포함.

규제 : 머신러닝 모델이 훈련 세트를 너무 과도하게 학습하지 못하도록 훼방하는 것.

⟶ 모델이 훈련세트에 과대적합되지 않도록 만듦.

⟶ 선형회귀 모델의 경우 특성에 곱해지는 계수 (or 기울기)의 크기를 작게 만듦.

* 선형회귀 모델에 규제를 추가한 모델을 '릿지' 와 '라쏘' 라고 부른다.

- 릿지 : 계수를 제곱한 값을 기준으로 규제 적용., 선형모델의 계수를 작게 만들어 과대적합을 완화시킨다. (비교적 효과가 좋아 널리 쓰임).
- 라쏘 : 계수의 절댓값을 기준으로 규제 적용., 릿지와 달리 계수 값을 아예 0으로 만들 수도 있다.

- 릿지 회귀 : 모든 객체를 만들고 fit() 메서드에서 훈련한 뒤 score() 메서드로 평가한다.
- 라쏘 회귀 : Ridge 클래스 ⟶ Lasso 클래스 변경.