

학습목표

- 지도 학습 알고리즘의 한 종류인 회귀 알고리즘에 대해 배웁니다.
- 다양한 선형 회귀 알고리즘의 장단점을 이해합니다.

Chapter

03

회귀 알고리즘과 모델 규제

놓어의 무게를 예측하라!

03-3

특성 공학과 규제

핵심 키워드

다중 회귀

특성 공학

릿지

라쏘

하이퍼파라미터

여러 특성을 사용한 다중 회귀에 대해 배우고 사이킷런의 여러 도구를 사용해 봅니다. 복잡한 모델의 과대적합을 막기 위한 릿지와 라쏘 회귀를 배웁니다.

시작하기 전에

혼공머신은 다항 회귀로 농어의 무게를 어느 정도 예측할 수 있지만, 여전히 훈련 세트보다 테스트 세트의 점수가 높은 점이 웬지 찝찝합니다. 이 문제를 해결하려면 제공보다 더 고차항을 넣어야 할 것 같은데 얼마큼 더 고차항을 넣어야 할지 모르고 수동으로 이렇게 고차항을 넣기도 힘듭니다. 혼공머신은 홍 선배에게 도움을 요청하기로 마음먹었습니다.

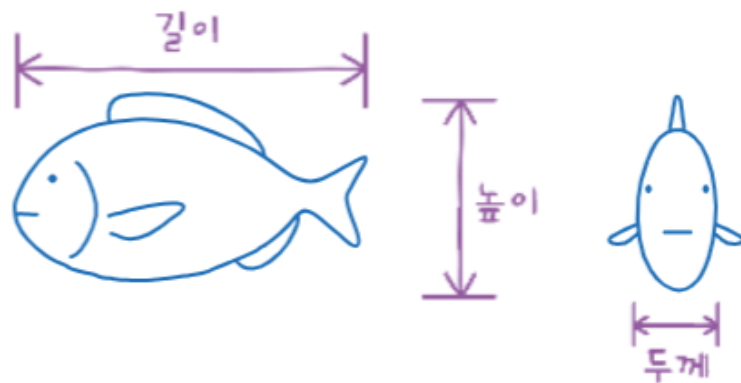
“농어의 무게를 예측하려고 몸통 길이와 깊이를 제공해서 선형 회귀 모델을 훈련시켰는데 여전히 과소적합인 것 같아요.”

“데이터가 농어의 길이뿐이야?”

“어... 사실 김 팀장님이 길이 말고 높이와 두께 데이터도 줬어요.”

“이런, 받은 데이터를 모두 사용해야지. 선형 회귀는 특성이 많을수록 엄청난 효과를 내거든. 높이와 두께를 다항 회귀에 함께 적용해 봐.

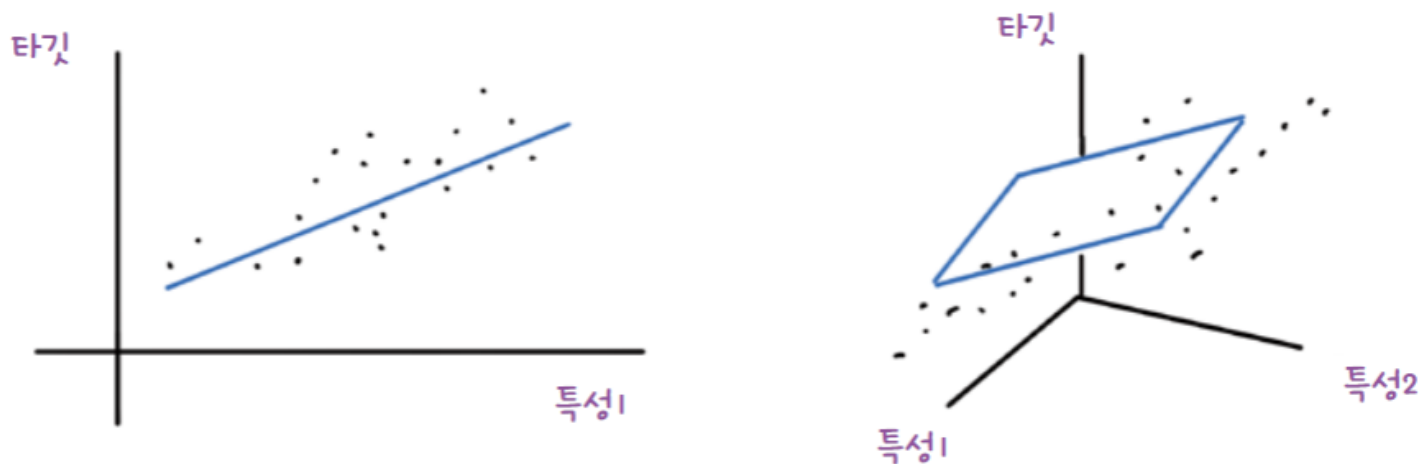
아, 수고스럽게 직접 만들지 말고 사이킷런의 PolynomialFeatures 클래스를 사용해. 훨씬 편하거든.”



다중 회귀

2절에서는 하나의 특성을 사용하여 선형 회귀 모델을 훈련시켰습니다. 여러 개의 특성을 사용한 선형 회귀를 다중 회귀 multiple regression라고 부릅니다.

2절에서처럼 1개의 특성을 사용했을 때 선형 회귀 모델이 학습하는 것은 직선입니다. 2개의 특성을 사용하면 무엇을 학습할까요? 특성이 2개면 선형 회귀는 평면을 학습합니다. 다음 그림에서 두 경우를 비교해 보세요.



오른쪽 그림처럼 특성이 2개면 타겟값과 함께 3차원 공간을 형성하고 선형 회귀 방정식 '타겟 = $a \times$ 특성1 + $b \times$ 특성2 + 절편'은 평면이 됩니다. 그럼 특성이 3개일 경우는 어떨까요?

이 예제에서는 농어의 길이뿐만 아니라 농어의 높이와 두께도 함께 사용하겠습니다. 또한 이전 절에서처럼 3개의 특성을 각각 제공하여 추가합니다. 거기다가 각 특성을 서로 곱해서 또 다른 특성을 만들겠습니다. 즉 ‘농어 길이 × 농어 높이’를 새로운 특성으로 만드는 거죠. 이렇게 기존의 특성을 사용해 새로운 특성을 뽑아내는 작업을 특성 공학 feature engineering이라고 부릅니다.

데이터 준비


이전과 달리 농어의 특성이 3개로 늘어났기 때문에 데이터를 복사해 붙여넣는 것도 번거롭습니다. 인터넷에서 데이터를 바로 다운로드하여 사용할 수는 없을까요? 아쉽지만 넘파이는 이런 작업을 잘 지원하지 않습니다. 하지만 판다스를 사용하면 아주 간단합니다.

판다스 pandas는 유명한 데이터 분석 라이브러리입니다. 데이터프레임 dataframe은 판다스의 핵심 데이터 구조입니다. 넘파이 배열과 비슷하게 다차원 배열을 다룰 수 있지만 훨씬 더 많은 기능을 제공하죠. 또 데이터프레임은 넘파이 배열로 쉽게 바꿀 수도 있습니다.

판다스를 사용해 놓어 데이터를 인터넷에서 내려받아 데이터프레임에 저장하겠습니다. 그다음 넘파이 배열로 변환하여 선형 회귀 모델을 훈련해 보죠. 판다스 데이터프레임을 만들기 위해 많이 사용하는 파일은 CSV 파일입니다. CSV 파일은 다음 그림처럼 콤마로 나누어져 있는 텍스트 파일입니다.

CSV 파일

length,	height,	width
8.4,	2.11,	1.41
13.7,	3.53,	2.0
	⋮	

손코딩

```
import pandas as pd # pd는 관례적으로 사용하는 판다스의 별칭입니다
df = pd.read_csv('https://bit.ly/perch_csv_data')
perch_full = df.to_numpy()
print(perch_full)
```

```
[[ 8.4   2.11  1.41]
 [13.7   3.53  2.  ]
 [15.    3.82  2.43]
 ...
 [44.   12.49  7.6 ]]
```



타깃 데이터는 이전과 동일한 방식으로 준비합니다.

note http://bit.ly/perch_data에서 복사해 쓰세요.

손코딩

```
import numpy as np
perch_weight = np.array(
    [ 5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0,
      110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0,
      130.0, 150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0,
      197.0, 218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0,
      514.0, 556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0,
      820.0, 850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0,
      1000.0, 1000.0]
)
```

그다음 `perch_full`과 `perch_weight`를 훈련 세트와 테스트 세트로 나눕니다.

손코딩

```
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

사이킷런은 특성을 만들거나 전처리하기 위한 다양한 클래스를 제공합니다. 사이킷런에서는 이런 클래스를 변환기^{transformer}라고 부릅니다. 사이킷런의 모델 클래스에 일관된 `fit()`, `score()`, `predict()` 메서드가 있는 것처럼 변환기 클래스는 모두 `fit()`, `transform()` 메서드를 제공합니다.

사이킷런에는 변환기라 부르는 특성을 만들거나 전처리하는 클래스가 있습니다.


note 앞서 배운 `LinearRegression` 같은 사이킷런의 모델 클래스는 추정기(estimator)라고도 부릅니다.

우리가 사용할 변환기는 `PolynomialFeatures` 클래스입니다. 먼저 이 클래스를 사용하는 방법을 알아보겠습니다. 이 클래스는 `sklearn.preprocessing` 패키지에 포함되어 있습니다.


손코딩

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

2개의 특성 2와 3으로 이루어진 샘플 하나를 적용하겠습니다. 앞서 이야기한 것처럼 이 클래스의 객체를 만든 다음 `fit()`, `transform()` 메서드를 차례대로 호출합니다.

 손코딩

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

 `[[1. 2. 3. 4. 6. 9.]]`

`fit ()` - 새롭게 만들 특성 조합을 찾음

`transform ()` - 실제 데이터 변환

✚ 여기서 잠깐

transform 전에 꼭 `poly.fit`을 사용해야 하나요?

훈련(`fit`)을 해야 변환(`transform`)이 가능합니다. 사이킷런의 일관된 api 때문에 두 단계로 나뉘어져 있습니다. 두 메서드를 하나로 붙인 `fit_transform` 메서드도 있습니다.

와우, 특성이 아주 많아졌군요! PolynomialFeatures 클래스는 기본적으로 각 특성을 제공한 항을 추가하고 특성끼리 서로 곱한 항을 추가합니다. 2와 3을 각기 제공한 4와 9가 추가되었고, 2와 3을 곱한 6이 추가되었습니다. 1은 왜 추가되었을까요? 다음의 식을 한번 보죠.

$$\text{무게} = a \times \text{길이} + b \times \text{높이} + c \times \text{두께} + d \times 1$$

사실 선형 방정식의 절편을 항상 값이 1인 특성과 곱해지는 계수라고 볼 수 있습니다. 이렇게 놓고 보면 특성은 (길이, 높이, 두께, 1)이 됩니다. 하지만 사이킷런의 선형 모델은 자동으로 절편을 추가하므로 굳이 이렇게 특성을 만들 필요가 없습니다. `include_bias=False`로 지정하여 다시 특성을 변환하겠습니다.

손코딩

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



```
[[2. 3. 4. 6. 9.]]
```

이제 이 방식으로 `train_input`에 적용하겠습니다. `train_input`을 변환한 데이터를 `train_poly`에 저장하고 이 배열의 크기를 확인해 보죠.

 손코딩

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



(42, 9)

`PolynomialFeatures` 클래스는 9개의 특성이 어떻게 만들어졌는지 확인하는 아주 좋은 방법을 제공합니다. 다음처럼 `get_feature_names_out()` 메서드를 호출하면 9개의 특성이 각각 어떤 입력의 조합으로 만들어졌는지 알려 줍니다.

 손코딩

```
poly.get_feature_names_out()
```



```
['x0', 'x1', 'x2', 'x0^2', 'x0 x1', 'x0 x2', 'x1^2', 'x1 x2', 'x2^2']
```

 손코딩

```
test_poly = poly.transform(test_input)
```

다중 회귀 모델 훈련하기

다시 한번 이야기하지만 다중 회귀 모델을 훈련하는 것은 선형 회귀 모델을 훈련하는 것과 같습니다. 다만 여러 개의 특성을 사용하여 선형 회귀를 수행하는 것뿐이죠. 먼저 사이킷런의 `LinearRegression` 클래스를 임포트하고 앞에서 만든 `train_poly`를 사용해 모델을 훈련시켜 보겠습니다.

 손코딩

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

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



```
0.9903183436982124
```

 손코딩

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



```
0.9714559911594134
```

농어의 길이, 높이, 두께를 사용.


각 특성을 제공하거나 서로 곱해서 다항 특성을 더 추가함.

과소적합 문제는 해결 됨!


특성을 더 많이 추가하면 어떨까요? 3제곱, 4제곱 항을 넣는 거죠. PolynomialFeatures 클래스의 degree 매개변수를 사용하여 필요한 고차항의 최대 차수를 지정할 수 있습니다. 5제곱까지 특성을 만들어 출력해 보겠습니다.

 손코딩

```
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)
```

 (42, 55) 손코딩

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

 0.9999999999991098

특성을 추가 할수록 모델은
훈련 세트에 대해서 완벽하
게 학습함

손코딩

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



```
-144.40579242684848
```

ㅋㅋㅋ 훈련 세트에 지나치게 과대적합되어서 테스트 세트에는 쓸 수가 없음.

+ 여기서 잠깐

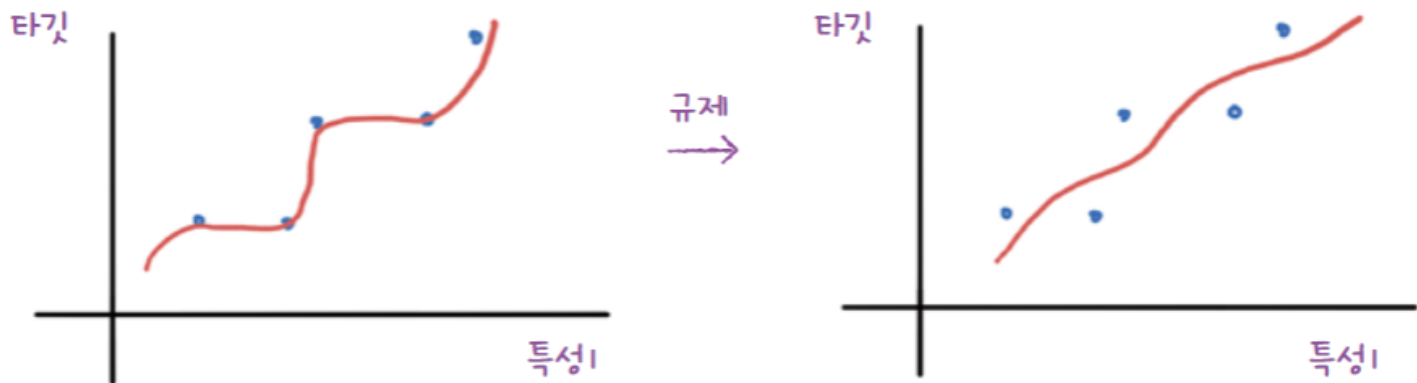
샘플 개수보다 특성이 많다면 어떨까요?

여기에서 사용한 훈련 세트의 샘플 개수는 42개 밖에 되지 않습니다. 42개의 샘플을 55개의 특성으로 훈련하면 완벽하게 학습할 수 있는 것이 당연합니다. 예를 들어 42개의 참새를 맞추기 위해 딱 한 번 새총을 쏘아 한다면 참새 떼 중앙을 겨냥하여 가능한 한 맞출 가능성을 높여야 합니다. 하지만 55번이나 쏠 수 있다면 한 번에 하나씩 모든 참새를 맞출 수 있습니다.

규제

규제 regularization는 머신러닝 모델이 훈련 세트를 너무 과도하게 학습하지 못하도록 꾀방하는 것을 말합니다. 즉 모델이 훈련 세트에 과대적합되지 않도록 만드는 것이죠. 선형 회귀 모델의 경우 특성에 곱해지는 계수(또는 기울기)의 크기를 작게 만드는 일입니다.

이해를 돕기 위해 다음 쪽과 같이 하나의 특성을 가진 데이터를 학습한 모델을 생각하겠습니다. 왼쪽은 훈련 세트를 과도하게 학습했고 오른쪽은 기울기를 줄여 보다 보편적인 패턴을 학습하고 있습니다.



손코딩

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_poly)
train_scaled = ss.transform(train_poly)
test_scaled = ss.transform(test_poly)
```

특성의 스케일이 정규화되지 않으면 당연히 특성에 곱해지는 계수 값도 달라지게 됨 - > 정규화를 해주어야 함! (여기서는 StandardScaler 를 사용해보자)

먼저 StandardScaler 클래스의 객체 ss를 초기화한 후 PolynomialFeatures 클래스로 만든 train_poly를 사용해 이 객체를 훈련합니다. 여기에서도 다시 한번 강조하지만 꼭 훈련 세트로 학습한 변환기를 사용해 테스트 세트까지 변환해야 합니다.

이제 표준점수로 변환한 train_scaled와 test_scaled가 준비되었습니다.

note 훈련 세트에서 학습한 평균과 표준편차는 StandardScaler 클래스 객체의 mean_, scale_ 속성에 저장됩니다. 특성마다 계산하므로 55개의 평균과 표준 편차가 들어 있습니다.

릿지 회귀

릿지와 라쏘 모두 `sklearn.linear_model` 패키지 안에 있습니다. 사이킷런 모델을 사용할 때 편리한 점은 훈련하고 사용하는 방법이 항상 같다는 것입니다. 모델 객체를 만들고 `fit()` 메서드에서 훈련한 다음 `score()` 메서드로 평가합니다. 앞서 준비한 `train_scaled` 데이터로 릿지 모델을 훈련해 보죠.

손코딩

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



0.9896101671037343

손코딩

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



0.9790693977615398

릿지(ridge)와 라쏘(lasso) 모델은 선형 회기 모델에 규제를 추가한 것임.

릿지는 계수를 제공한 값을 기준으로 규제를 적용.

라쏘는 계수의 절댓값을 기준으로 규제를 적용.

+ 여기서 잠깐

사람이 직접 지정해야 하는 매개변수

alpha 값은 릿지 모델이 학습하는 값이 아니라 사전에 우리가 지정해야 하는 값입니다. 이렇게 머신러닝 모델이 학습할 수 없고 사람이 알려줘야 하는 파라미터를 **하이퍼파라미터(hyperparameter)**라고 부릅니다. 사이킷런과 같은 머신러닝 라이브러리에서 하이퍼파라미터는 클래스와 메서드의 매개변수로 표현됩니다. 이 책에서는 함수와 클래스의 파라미터는 매개변수라고 하고 모델과 관련된 파라미터(모델 파라미터, 하이퍼파라미터)는 그대로 파라미터라고 표현했습니다.

규제의 강조를 조절하는 alpha 매개변수가 있음. 값이 커질수록 규제의 강도가 세져서 계수 값을 더 줄이고 과소적합 되도록 유도함.

적절한 alpha 값을 찾는 한 가지 방법은 alpha 값에 대한 R^2 값의 그래프를 그려 보는 것입니다. 훈련 세트와 테스트 세트의 점수가 가장 가까운 지점이 최적의 alpha 값이 됩니다. 먼저 맷플롯립을 임포트하고 alpha 값을 바꿀 때마다 `score()` 메서드의 결과를 저장할 리스트를 만듭니다.

손코딩

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

다음 코드는 alpha 값을 0.001에서 100까지 10배씩 늘려가며 릿지 회귀 모델을 훈련한 다음 훈련 세트와 테스트 세트의 점수를 파이썬 리스트에 저장합니다.

손코딩

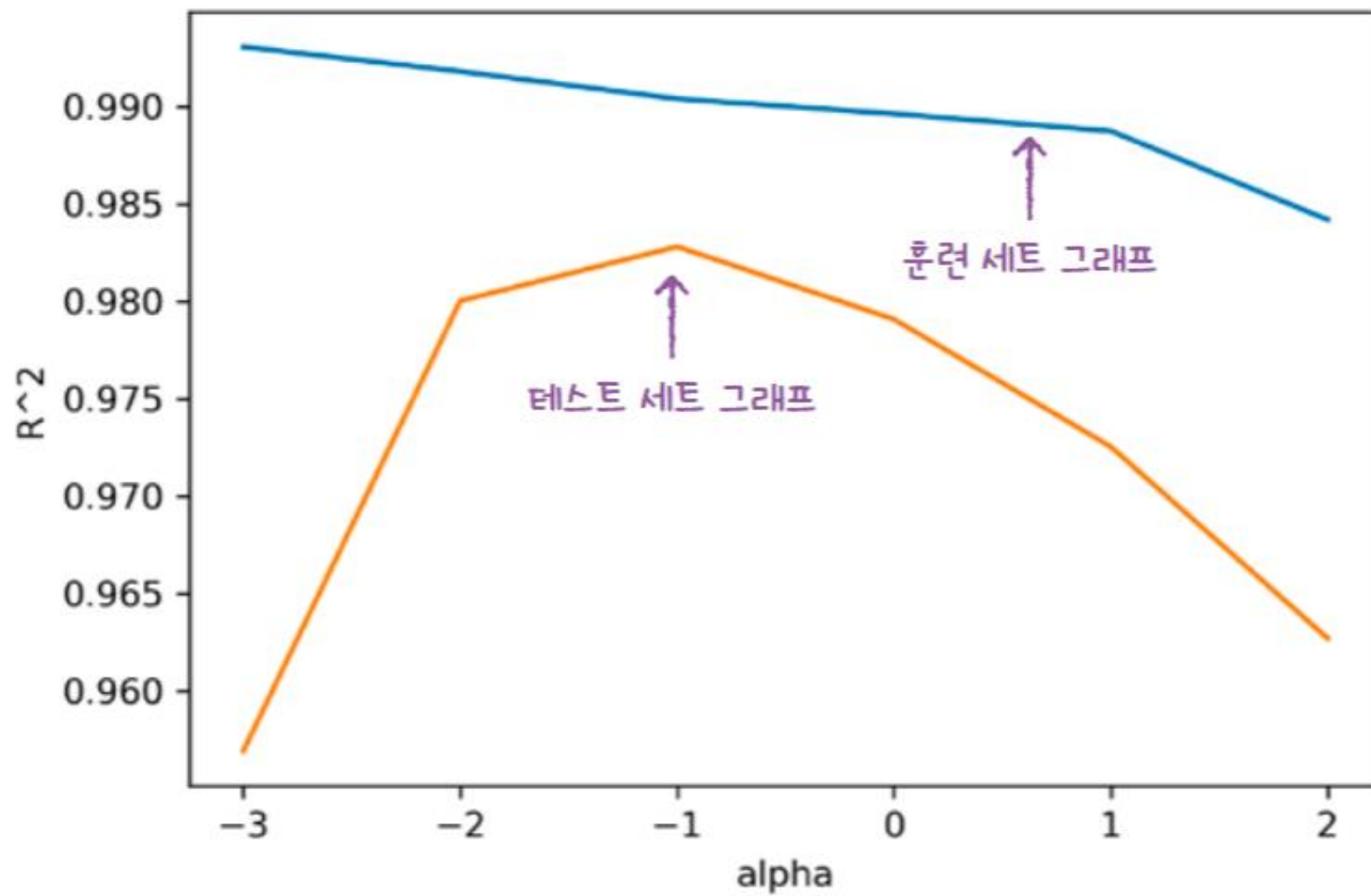
```
alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]  
for alpha in alpha_list:  
    # 릿지 모델을 만듭니다  
    ridge = Ridge(alpha=alpha)  
    # 릿지 모델을 훈련합니다  
    ridge.fit(train_scaled, train_target)  
    # 훈련 점수와 테스트 점수를 저장합니다  
    train_score.append(ridge.score(train_scaled, train_target))  
    test_score.append(ridge.score(test_scaled,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


이제 그래프를 그려보겠습니다. α 값을 0.001부터 10배씩 늘렸기 때문에 이대로 그래프를 그리면 그래프 왼쪽이 너무 촘촘해집니다. `alpha_list`에 있는 6개의 값을 동일한 간격으로 나타내기 위해 로그 함수로 바꾸어 지수로 표현하겠습니다. 즉 0.001은 -3 , 0.01은 -2 가 되는 식입니다.

note 넘파이 로그 함수는 `np.log()`와 `np.log10()`이 있습니다. 전자는 자연 상수 e 를 밑으로 하는 자연로그입니다. 후자는 10을 밑으로 하는 상용로그입니다.

손코딩

```
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.show()
```



손코딩

```
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))
```



0.9903815817570366

0.9827976465386926

라쏘 회귀

라쏘 모델을 훈련하는 것은 릿지와 매우 비슷합니다. Ridge 클래스를 Lasso 클래스로 바꾸는 것이 전부입니다.

손코딩

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



0.9897898972080961

손코딩

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



0.9800593698421883

손코딩

```
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))
```


+ 여기서 잠깐

경고(Warning)가 뜹니다. 정상인가요?

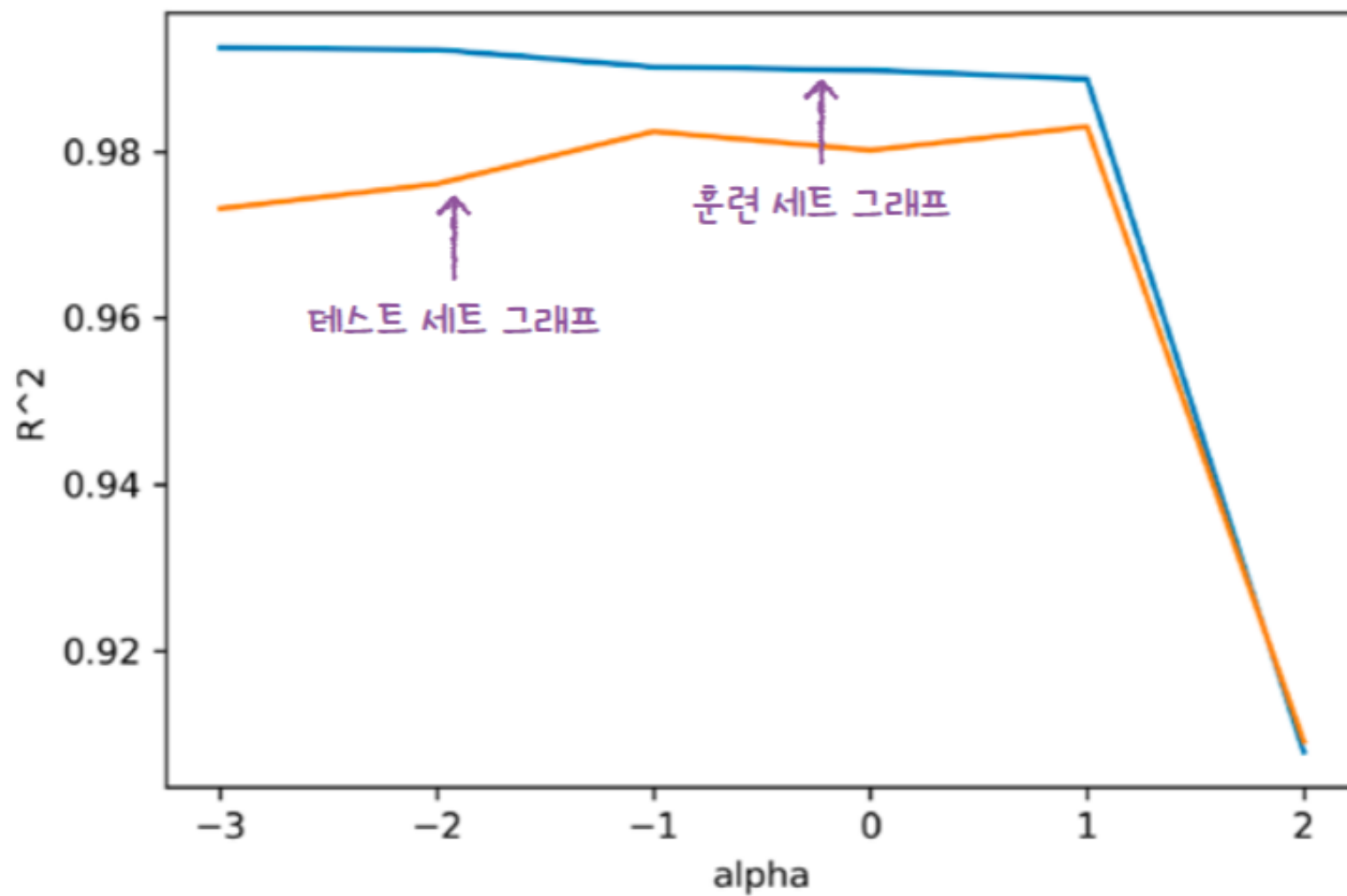
```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.  
py:476: ConvergenceWarning: Objective did not converge. You might want to  
increase the number of iterations. Duality gap: 18778.697957792876, tolerance:  
518.2793833333334 positive)
```

라쏘 모델을 훈련할 때 ConvergenceWarning이란 경고가 발생할 수 있습니다. 사이킷런의 라쏘 모델은 최적의 계수를 찾기 위해 반복적인 계산을 수행하는데, 지정한 반복 횟수가 부족할 때 이런 경고가 발생합니다. 이 반복 횟수를 충분히 늘리기 위해 max_iter 매개변수의 값을 10000으로 지정했습니다. 필요하다면 더 늘릴 수 있지만 이 문제에서는 큰 영향을 끼치지 않습니다.

그다음 `train_score`와 `test_score` 리스트를 사용해 그래프를 그립니다. 이 그래프도 x축은 로그 스케일로 바꿔 그리겠습니다.

손코딩

```
plt.plot(np.log10(alpha_list), train_score)
plt.plot(np.log10(alpha_list), test_score)
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.show()
```



손코딩

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



```
0.9888067471131867
0.9824470598706695
```

손코딩

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



```
40
```

55개의 특성을 사용했지만,
라쏘모델이 사용한 특성은 15
개 뿐임.

note

`np.sum()` 함수는 배열을 모두 더한 값을 반환합니다. 넘파이 배열에 비교 연산자를 사용했을 때 각 원소는 True 또는 False가 됩니다. `np.sum()` 함수는 True를 1로, False를 0으로 인식하여 덧셈을 할 수 있기 때문에 마치 비교 연산자에 맞는 원소 개수를 헤아리는 효과를 냅니다.

▶ 키워드로 끝내는 핵심 포인트

- **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 α 파라미터입니다.

▶ 핵심 패키지와 함수

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.

`sep`는 CSV 파일의 구분자를 지정합니다. 기본값은 ‘coma(,)’입니다.

`header`에 데이터프레임의 열 이름으로 사용할 CSV 파일의 행 번호를 지정합니다. 기본적으로 첫 번째 행을 열 이름으로 사용합니다.

`skiprows`는 파일에서 읽기 전에 건너뛸 행의 개수를 지정합니다.

`nrows`는 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 때에 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.

▶ 확인 문제

1. a, b, c 특성으로 이루어진 훈련 세트를 PolynomialFeatures(degree=3)으로 변환했습니다. 다음 중 이 변환된 데이터에 포함되지 않는 특성은 무엇인가요?

- ① 1
- ② a
- ③ $a * b$
- ④ $a * b^3$

2. 다음 중 특성을 표준화하는 사이킷런 변환기 클래스는 무엇인가요?

- ① Ridge
- ② Lasso
- ③ StandardScaler
- ④ LinearRegression

3. 다음 중 과대적합과 과소적합을 올바르게 표현하지 못한 것은 무엇인가요?

- ① 과대적합인 모델은 훈련 세트의 점수가 높습니다.
- ② 과대적합인 모델은 테스트 세트의 점수도 높습니다.
- ③ 과소적합인 모델은 훈련 세트의 점수가 낮습니다.
- ④ 과소적합인 모델은 테스트 세트의 점수도 낮습니다.

축하합니다! 여러분은 열심히 공부해서 갓G 전자의 마케팅 부서로 입사했습니다!

그 중 TV에 사용하는 display 팀에 속한 여러분은, 이번 분기에 필요한 예산을 본부에 요청하려고 합니다.

물론 예산은 많을수록 좋겠지만, 본부는 최근 경기 침체 등을 이유로 예산을 최대한 삭감하려고 합니다.

따라서 여러분은 기존 자료에서 마케팅 예산에 따른 매출을 예측하고자 합니다.

그리고보니 3학년 머신러닝 시간에 배웠던 선형 회귀 모델이 비슷한 역할을 했던 것 같습니다.

우리가 아직 배우지 않은 코드는 제가 미리 입력 해두었습니다.

Question.

주어진 데이터 셋의 산점도를 그리세요.

주어진 데이터 셋으로 변수가 1개인 linear model을 만들고 intercept and coefficients를 구하세요.

변수가 1개인 linear model의 performance score를 구하고 해당 모델이 과대 or 과소 적합인지 구하세요.

변수가 2개인 linear model의 performance score를 구하고 해당 모델이 과대 or 과소 적합인지 구하세요.

pandas import

```
import pandas as pd
```

numpy import

```
import numpy as np
```

read csv file from github

```
url = "https://raw.githubusercontent.com/devzohaib/Simple-Linear-Regression/master/tvmarketing.csv"
```

```
advertising = pd.read_csv(url)
```


check the shape of the dataframe

```
advertising.shape
```

check some statistical information

```
advertising.describe()
```

prepare input and target

```
input = np.array(advertising['TV'])
```

```
target = np.array(advertising['Sales'])
```

visualize the relationship between TV (on the x-axis) and Sales (on the y-axis)



#use train_test_split function to split data into training and testing samples. Use default split ratio.



#check the shape of input_train, input_test, target_train, target_test



import LinearRegression from sklearn



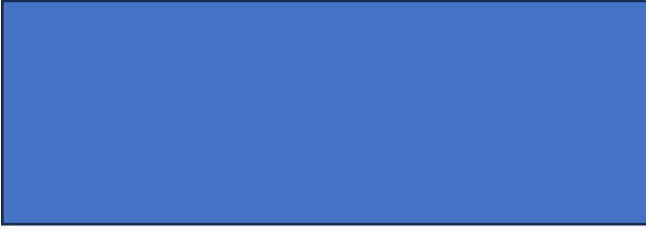
Representing LinearRegression as lr(Creating LinearRegression Object)



Fit the model using lr.fit()



Print the intercept and coefficients



calculate the score and check if your model is over- or underfitted



prepare the data for polynomial having quadratic equation



train your poly-model and check if your model is over- or underfitted

