인공신경망과딥러닝입문

Lecture 08. 특성 공학과 규제

동덕여자대학교
데이터사이언스 전공
권 범

목차

01. 다중 회귀

02. 사이킷런의 변환기
03. 다중 회귀 모델 훈련하기
04. 규제
05. 모델의 과대적합을 제어하기
06. 마무리

01. 다중 회귀

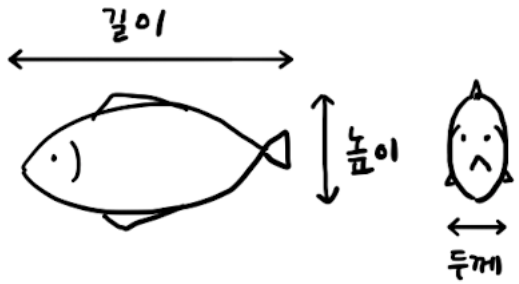
❖ 시작하기 전에

상황 가정

- ✓ 다항 회귀로 농어의 무게를 어느 정도 예측할 수 있지만,
  여전히 학습 데이터셋보다 시험 데이터셋의 점수가 높은 점이 왠지 찜찜함
- ✓ 이 문제를 해결하려면 제곱보다 더 고차항을 넣어야 할 것 같은데,
  얼만큼 더 고차항을 넣어야 할지 모르고 수동으로 이렇게 고차항을 넣기도 힘듦

선형 회귀는 특성이 많을수록 엄청난 효과를 냄.
이번에는 높이와 두께를 다항 회귀에 함께 적용해 보자

수고스럽게 직접 만들지 말고, scikit-learn의
PolynomialFeatures 클래스를 사용해 보자



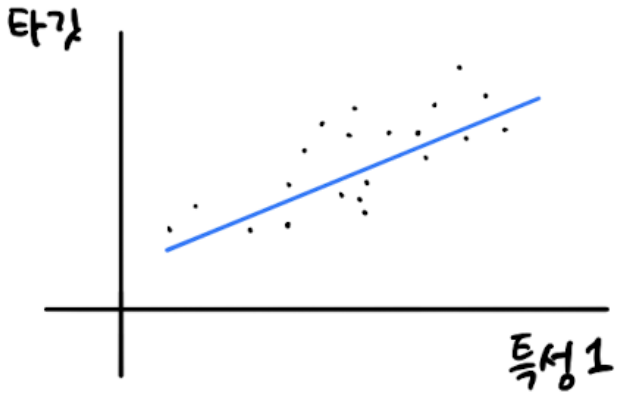
[사진출처] 혼자 공부하는 머신러닝+딥러닝 (출판사: 한빛아카데미)

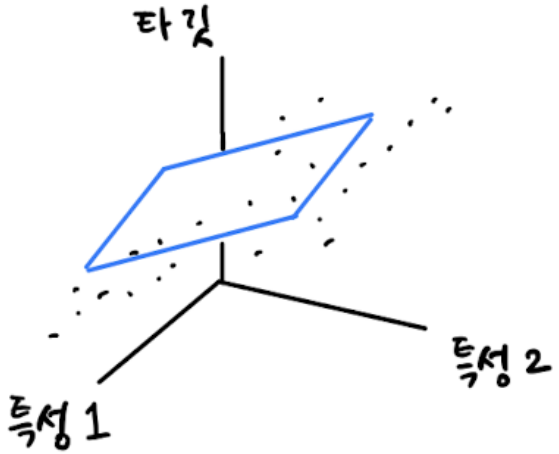
01. 다중 회귀

❖ 다중 회귀(Multiple Regression)란? (1/4)

- 이전 수업에서는 하나의 특성(=길이)를 사용하여 선형 회귀 모델을 학습시켰음
- 여러 개의 특성을 사용한 선형 회귀를 **다중 회귀**라고 부름

✓ 1개의 특성을 사용했을 때 선형 회귀 모델이 학습하는 것은 직선임
✓ 2개의 특성을 사용하면 무엇을 학습할까?
✓ 특성이 2개면 선형 회귀는 평면을 학습함
✓ 다음 그림에서 두 경우를 비교해 보자





[사진출처] 혼자 공부하는 머신러닝+딥러닝 (출판사: 한빛아카데미)

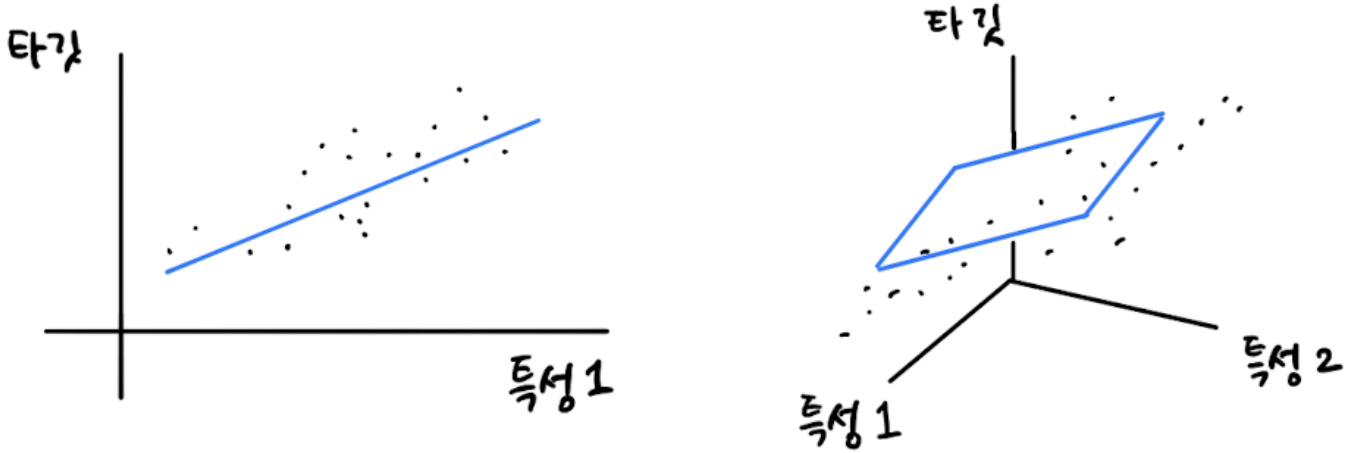
01. 다중 회귀

❖ 다중 회귀(Multiple Regression)란? (2/4)

- 오른쪽 그림처럼 특성이 2개면 타겟(=레이블) 값과 함께 3차원 공간을 형성하고, 선형 회귀 방정식은 평면이 됨

$$\text{타겟} = a \times \text{특성}1 + b \times \text{특성}2 + \text{절편}$$

그럼 특성이 3개인 경우는 어떨까?



[사진출처] 혼자 공부하는 머신러닝+딥러닝 (출판사: 한빛아카데미)

01. 다중 회귀

❖ 다중 회귀(Multiple Regression)란? (3/4)

- 안타깝지만 우리는 3차원 공간 이상을 그리거나 상상할 수 없음
- 분명한 것은 선형 회귀를 단순한 직선이나 평면으로 생각하여 성능이 무조건 낮다고 오해해서는 안됨

특성이 많은 고차원에서는 선형 회귀가 매우 복잡한 모델을 표현할 수 있음

01. 다중 회귀

❖ 다중 회귀(Multiple Regression)란? (4/4)

- 이번 수업에서는 농어의 길이뿐만 아니라 농어의 높이와 두께도 함께 사용해 보자
- 또한 이전 수업에서처럼 3개의 특성을 각각 제곱하여 추가해 보자
- 거기다가 각 특성을 서로 곱해서 또 다른 특성을 만들어 보자
- 즉, '농어 길이×농어 높이'를 새로운 특성으로 만들어 보자

“특성 공학”
(Feature Engineering)

이렇게 기존의 특성을 사용해 새로운 특성을 뽑아내는 작업을 특성 공학이라고 부름

우리가 직접 특성을 제곱하고 특성끼리 곱해서 새로운 특성을 추가할 수도 있지만, scikit-learn에서 제공하는 편리한 도구를 사용하겠음

01. 다중 회귀

❖ 데이터 준비 (1/10)

- 이전과 달리 다루는 특성의 개수가 3개로 늘어났기 때문에, 특성을 추출해 내는 것이 번거로움

```
1  import csv
2
3  f = open("Fish.csv", 'r')
4  data = csv.reader(f)
5
6  header = next(data)
7
8  perch_weight = []        # 농어의 무게(=레이블, 타겟)
9  perch_length = []        # 농어의 길이
10 perch_height = []        # 농어의 높이
11 perch_width = []         # 농어의 두께
12 for row in data:
13   if row[0] == "Perch":  # 농어
14     perch_weight.append(row[1])
15     perch_length.append(row[3])
16     perch_height.append(row[5])
17     perch_width.append(row[6])
18
19 f.close()
```

보다 편리한 방법은 없을까?

01. 다중 회귀

❖ 데이터 준비 (2/10)

- 판다스(pandas)를 사용하면 아주 간단함

“판다스”

(pandas)

데이터를 처리하고, 분석할 때 사용하는 라이브러리임

✓ 데이터프레임(DataFrame)은 판다스의 핵심 데이터 구조임
✓ 넘파이 배열과 비슷하게, 다차원 배열을 다룰 수 있지만 훨씬 더 많은 기능을 제공함
✓ 또한, 데이터프레임은 넘파이 배열로 쉽게 바꿀 수 있음

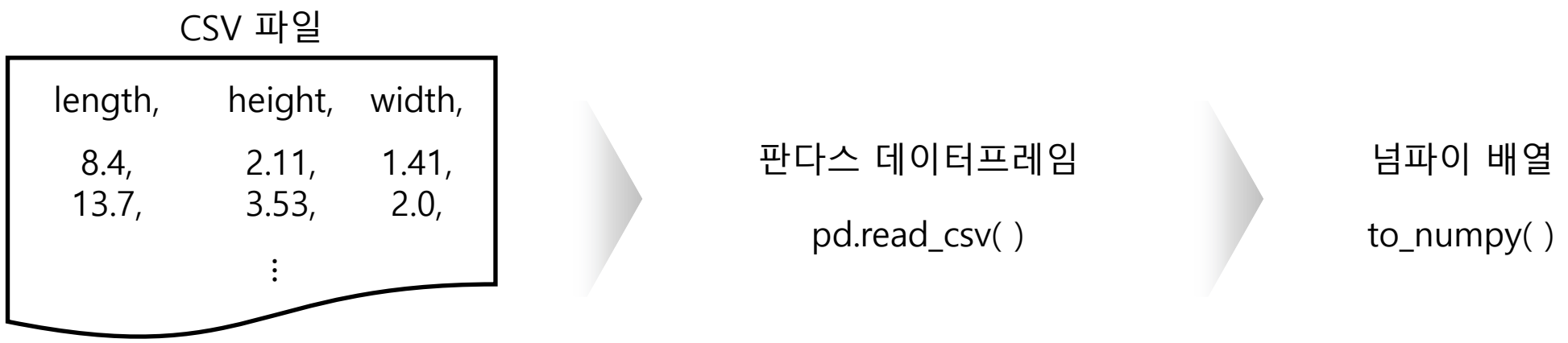
[사진출처] https://en.wikipedia.org/wiki/Pandas_(software)#/media/File:Pandas_logo.svg

01. 다중 회귀

❖ 데이터 준비 (3/10)

- 판다스를 사용해서 농어 데이터를 데이터프레임에 저장하자
- 그다음 넘파이 배열로 변환하여 선형 회귀 모델을 학습시켜 보자

어떻게 하면 되는지 살펴보자!



✓ CSV 파일을 판다스에서 읽는 방법은 아주 간단함
✓ 판다스의 read_csv( ) 함수에 파일의 이름(주소)을 넣어 주는 것이 전부임
✓ read_csv( ) 함수로 데이터프레임을 만든 다음, to_numpy( ) 메서드를 사용해 넘파이 배열로 바꿈

01. 다중 회귀

❖ 데이터 준비 (4/10)

- pandas 라이브러리를 임포트하고 'pd'라는 별명으로 부르자
- 판다스 read_csv( ) 함수에 파일의 이름(주소)을 전달하면 간단하게 파일을 읽을 수 있음

```
1 import pandas as pd       # pd는 관례적으로 사용하는 판다스의 별칭임
2
3 df = pd.read_csv("Fish.csv")
```

✓ 구글 코랩과 주피터랩에는 판다스가 이미 준비되어 있음
✓ 판다스를 임포트 할 때는 관례적으로 'pd'라는 별칭을 사용함

- import
  (명사) 수입(품)
  (동사) 수입하다
  (동사) 불러오다
- alias
  (부사) ~라는 가명으로 알려진
  (부사) 일명 ~라 불리는

01. 다중 회귀

❖ 데이터 준비 (5/10)

- df 데이터프레임에 저장된 내용을 살펴보자

```
1 print(type(df))
2 df
```

실행결과

<class 'pandas.core.frame.DataFrame'>

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340
...	...	...	...	...	...	...	...
154	Smelt	12.2	11.5	12.2	13.4	2.0904	1.3936
155	Smelt	13.4	11.7	12.4	13.5	2.4300	1.2690
156	Smelt	12.2	12.1	13.0	13.8	2.2770	1.2558
157	Smelt	19.7	13.2	14.3	15.2	2.8728	2.0672
158	Smelt	19.9	13.8	15.0	16.2	2.9322	1.8792

159 rows × 7 columns

Fish.csv 파일에는 총 159개의 샘플이 존재함!

01. 다중 회귀

❖ 데이터 준비 (6/10)

- 농어 데이터만 추출해 보자

```
1 df = df[df[“Species”] == “Perch”]
2 df
```

실행결과

	Species	Weight	Length1	Length2	Length3	Height	Width
72	Perch	5.9	7.5	8.4	8.8	2.1120	1.4080
73	Perch	32.0	12.5	13.7	14.7	3.5280	1.9992
74	Perch	40.0	13.8	15.0	16.0	3.8240	2.4320
75	Perch	51.5	15.0	16.2	17.2	4.5924	2.6316
76	Perch	70.0	15.7	17.4	18.5	4.5880	2.9415
77	Perch	100.0	16.2	18.0	19.2	5.2224	3.3216
78	Perch	78.0	16.8	18.7	19.4	5.1992	3.1234
79	Perch	80.0	17.2	19.0	20.2	5.6358	3.0502
80	Perch	85.0	17.8	19.6	20.8	5.1376	3.0368

⋮

대괄호 [ ] 안에 조건을 적으면 됨.
조건을 만족하는 행만 추출할 수 있음

01. 다중 회귀

❖ 데이터 준비 (7/10)

- 특성으로 사용할 길이(Length2), 높이(Height), 두께(Width) 열을 추출하자
- 그리고 레이블(타겟)으로 사용할 무게(Weight) 열도 추출하자

```
1 df_features = df[[“Length2”, “Height”, “Width”]]          # 특성 3개
2 df_label = df[[“Weight”]]                                 # 레이블(타겟)
```

	Length2	Height	Width
72	8.4	2.1120	1.4080
73	13.7	3.5280	1.9992
74	15.0	3.8240	2.4320
75	16.2	4.5924	2.6316
76	17.4	4.5880	2.9415
77	18.0	5.2224	3.3216
78	18.7	5.1992	3.1234
79	19.0	5.6358	3.0502
⋮			

	Weight
72	5.9
73	32.0
74	40.0
75	51.5
76	70.0
77	100.0
78	78.0
79	80.0
⋮	

대괄호 [ ] 안에 조건을 적으면 됨.
조건을 만족하는 행만 추출할 수 있음

01. 다중 회귀

❖ 데이터 준비 (8/10)

- 새롭게 반환된 df_features 데이터프레임을 to_numpy( ) 메서드로 넘파이 배열로 바꿈

```
1 perch_full = df_features.to_numpy()
2 print(perch_full)
```

실행결과

```
[[ 8.4   2.112   1.408 ]
 [13.7   3.528   1.9992]
 [15.    3.824   2.432 ]
 [16.2   4.5924  2.6316]
 [17.4   4.588   2.9415]
 [18.    5.2224  3.3216]
 … (중략) …
 [40.    11.135  6.63 ]
 [42.    12.8002 6.8684]
 [43.    11.9328 7.2772]
 [43.    12.5125 7.4165]
 [43.5   12.604  8.142 ]
 [44.    12.4888 7.5958]]
```

01. 다중 회귀

❖ 데이터 준비 (9/10)

- 새롭게 반환된 df_label 데이터프레임도 to_numpy( ) 메서드로 넘파이 배열로 바꿈

```
1 perch_weight = df_label.to_numpy()
2 print(perch_weight)
```

실행결과

```
[[ 5.9  ]
 [ 32.  ]
 [ 40.  ]
 [ 51.5 ]
 [ 70.  ]
 [ 100. ]
 … (중략) …
 [ 820. ]
 [1100. ]
 [1000. ]
 [1100. ]
 [1000. ]
 [1000. ]]
```

01. 다중 회귀

❖ 데이터 준비 (10/10)

- 그다음 perch_full과 perch_weight를 학습 데이터셋과 시험 데이터셋으로 나눔

```
1 from sklearn.model_selection import train_test_split
2
3 x_train, x_test, y_train, y_test = train_test_split(perch_full, perch_weight,
4                                                     random_state=42)
```

이 데이터를 사용해 새로운 특성을 만들어 보자!

02. 사이킷런의 변환기

01. 다중 회귀
03. 다중 회귀 모델 훈련하기
04. 규제
05. 모델의 과대적합을 제어하기
06. 마무리

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (1/8)

- scikit-learn은 특성을 만들거나 전처리(Preprocessing)하기 위한 다양한 클래스를 제공함
- scikit-learn에서는 이런 클래스를 변환기(Transformer)라고 부름
- scikit-learn의 모델 클래스에 일관된 fit( ), score( ), predict( ) 메서드가 있는 것처럼
  변환기 클래스는 모두 fit( ), transform( ) 메서드를 제공함
- 앞서 배운 LinearRegression 같은 scikit-learn의 모델 클래스는 추정기(Estimator)라고도 불림

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (2/8)

- 우리가 사용할 변환기는 PolynomialFeatures 클래스임
- 먼저 이 클래스를 사용하는 방법을 알아보자

```
1 from sklearn.preprocessing import PolynomialFeatures
2
3 poly = PolynomialFeatures()
4 poly.fit([[2, 3]])
5 print(poly.transform([[2, 3]]))
```

sklearn.preprocessing 패키지에 포함되어 있음

✓ 2개의 특성 2와 3으로 이루어진 샘플 하나를 적용해 보자
✓ 앞서 이야기한 것처럼 이 클래스의 객체를 만든 다음, fit( ), transform( ) 메서드를 차례대로 호출하면 됨

실행결과

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

✓ fit( ) 메서드는 새롭게 만들 특성 조합을 찾음
✓ transform( ) 메서드는 실제로 데이터를 변환함
✓ 변환기는 입력 데이터를 변환하는 데 레이블 데이터를 필요로 하지 않음
✓ 따라서, 모델 클래스와는 다르게 fit( ) 메서드에 입력 데이터만 전달함
✓ 즉, 여기에서는 2개의 특성 (원소)을 가진 샘플 [2, 3]이, 6개의 특성을 가진 샘플 [1. 2. 3. 4. 6. 9.]로 바뀜

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (3/8)

transform 전에 꼭 poly.fit을 사용해야 하나요?

- ✓ 학습(fit)을 해야 변환(transform)이 가능하기 때문에, 사용해야 함
- ✓ scikit-learn의 일관된 API 때문에 두 단계로 나뉘어져 있음
- ✓ 두 메서드를 하나로 붙인 fit_transform 메서드도 있음

❖ PolynomialFeatures 클래스 (4/8)

- PolynomialFeatures 클래스는 기본적으로 각 특성을 제곱한 항을 추가하고, 특성끼리 서로 곱한 항을 추가함

  ✓ 2와 3을 각기 제곱한 4와 9가 추가됨
  ✓ 그리고 2와 3을 곱한 6이 추가됨

- 1은 왜 추가되었을까? 아래의 식을 살펴보자

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

  ✓ 사실 선형 방정식의 절편을, 항상 값이 1인 특성과 곱해지는 계수라고 볼 수 있음
  ✓ 이렇게 놓고 보면 특성은 (길이, 높이, 두께, 1)이 됨
  ✓ 하지만 scikit-learn의 선형 모델은 자동으로 절편을 추가하므로, 굳이 이렇게 특성을 만들 필요가 없음

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (5/8)

- include_bias=False로 지정하여, 다시 특성을 변환하자

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

실행결과

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

절편을 위한 항이 제거되고,
특성의 제곱과 특성끼리 곱한 항만 추가되었음

include_bias=False는 꼭 지정해야 하나요?

✓ scikit-learn 모델은 자동으로 특성에 추가된 절편 항을 무시하기 때문에, 지정하지 않아도 됨
✓ 하지만 본 수업에서는, 혼돈을 피하기 위해 명시적으로 지정하였음

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (6/8)

- 이제 이 방식을 x_train에 적용하자
- x_train을 변환한 데이터를 x_train_poly에 저장하고, 이 배열의 크기를 확인해 보자

```
1 poly = PolynomialFeatures(include_bias=False)
2 poly.fit(x_train)
3 x_train_poly = poly.transform(x_train)
4 print(x_train_poly.shape)
```

실행결과

```
(42, 9)
```

특성의 개수가 9가 되었음!

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (7/8)

- PolynomialFeatures 클래스는 9개의 특성이 어떻게 만들어졌는지 확인하는 방법을 제공함
- 다음처럼 get_feature_names_out( ) 메서드를 호출하면
  9개의 특성이 각각 어떤 입력의 조합으로 만들어졌는지 알려 줌

```
1 poly.get_feature_names_out()
```

실행결과

```
array(['x0', 'x1', 'x2', 'x0^2', 'x0 x1', 'x0 x2', 'x1^2', 'x1 x2',
       'x2^2'], dtype=object)
```

✓ 'x0': 첫 번째 특성을 의미
✓ 'x0^2': 첫 번째 특성의 제곱을 의미
✓ 'x0 x1': 첫 번째 특성과 두 번째 특성의 곱을 의미

02. 사이킷런의 변환기

❖ PolynomialFeatures 클래스 (8/8)

- 이제 시험 데이터셋을 변환하자

```
1 x_test_poly = poly.transform(x_test)
```

꼭 학습 데이터셋에 적용했던 변환기로 시험 데이터셋을 변환해야 하나요?

- ✓ 사실 PolynomialFeatures 클래스는 fit( ) 메서드에서 만들 특성의 조합을 준비하기만 하고 별도의 통계 값을 구하지 않음
- ✓ 따라서 시험 데이터셋을 따로 변환해도 됨
- ✓ 하지만 항상 학습 데이터셋을 기준으로 시험 데이터셋을 변환하는 습관을 들이는 것이 좋음

이어서 변환된 특성을 사용하여 다중 회귀 모델을 학습시켜 보자!

03. 다중 회귀 모델 훈련하기

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (1/7)

- 다중 회귀 모델을 훈련하는 것은 선형 회귀 모델을 훈련하는 것과 같음
- 다만 여러 개의 특성을 사용하여 선형 회귀를 수행하는 것뿐임
- 먼저 scikit-learn의 LinearRegression 클래스를 임포트하고,
  앞에서 만든 x_train_poly를 사용해 모델을 훈련시켜 보자

```
1 from sklearn.linear_model import LinearRegression
2
3 lr = LinearRegression()
4 lr.fit(x_train_poly, y_train)
5
6 print(lr.score(x_train_poly, y_train))
```

실행결과

```
0.9903557670312702
```

특성이 늘어나면, 선형 회귀의 능력은 매우 강력해진다는 것을 알 수 있음

✓ 학습 데이터셋에 대해서 높은 점수가 나왔음
✓ 농어의 길이뿐만 아니라 높이와 두께를 모두 사용했고,
각 특성을 제곱하거나 서로 곱해서 다항 특성을 추가했음

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (2/7)

- 시험 데이터셋에 대한 점수도 확인해 보자

```
1 print(lr.score(x_test_poly, y_test))
```

실행결과

```
0.9712376207461857
```

✓ 시험 데이터셋에 대한 점수는 높아지지 않았음
✓ 하지만, 농어의 길이만 사용했을 때 있던 과소적합 문제는 더이상 나타나지 않음

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (3/7)

- 특성을 더 많이 추가하면 어떨까? 3제곱, 4제곱 항을 넣어 보자
- PolynomialFeatures 클래스의 degree 매개변수를 사용하여 필요한 고차항의 최대 차수를 지정할 수 있음
- 5제곱까지 특성을 만들어 출력해 보자

```
1 poly = PolynomialFeatures(degree=5, include_bias=False)
2 poly.fit(x_train)
3 x_train_poly = poly.transform(x_train)
4 x_test_poly = poly.transform(x_test)
5 print(x_train_poly.shape)
```

실행결과

```
(42, 55)
```

만들어진 특성의 개수가 무려 55개나 됨!

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (4/7)

- 이 데이터를 사용해 선형 회귀 모델을 다시 훈련하자

```
1 lr.fit(x_train_poly, y_train)
2 print(lr.score(x_train_poly, y_train))
```

실행결과

```
0.9999999999996642
```

거의 완벽한 점수임!

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (5/7)

- 시험 데이터셋에 대한 점수를 확인해 보자

```
1 print(lr.score(x_test_poly, y_test))
```

실행결과

```
-129.88151246056472
```

아주 큰 음수임!

이게 무슨 일이죠?
문제가 무엇인지 눈치챘나요?

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (6/7)

- 특성의 개수를 크게 늘리면 선형 모델은 아주 강력해 짐
- 학습 데이터셋에 대해 거의 완벽하게 학습할 수 있음
- 하지만 이런 모델은 학습 데이터셋에 과대적합되므로, 시험 데이터셋에서는 형편없는 점수를 만듦

03. 다중 회귀 모델 훈련하기

❖ 모델 훈련 (7/7)

샘플 개수보다 특성이 많다면 어떨까?

- ✓ 우리가 사용한 학습 데이터셋의 샘플 개수는 42개 밖에 되지 않음
- ✓ 42개의 샘플을 55개의 특성으로 훈련하면 완벽하게 학습할 수 있는 것이 당연함
- ✓ 예를 들어, 42개의 참새를 맞추기 위해 딱 한 번 새총을 쏴야 한다면 참새 떼 중앙을 겨냥하여 가능한 맞출 가능성을 높여야 함
- ✓ 하지만 55번이나 쏠 수 있다면 한 번에 하나씩 모든 참새를 맞출 수 있음

이러한 문제를 차원의 저주(Curse of Dimensionality)라고 하며, 차원의 저주 문제를 해결하려면 다시 특성의 개수를 줄여야 함

하지만 이런 상황은 과대적합을 줄이는 또 다른 방법을 배워 볼 좋은 기회임

04. 규제

01. 다중 회귀
02. 사이킷런의 변환기
03. 다중 회귀 모델 훈련하기
05. 모델의 과대적합을 제어하기
06. 마무리

04. 규제

❖ 규제 이해하기 (1/2)

“규제”
(Regularization)

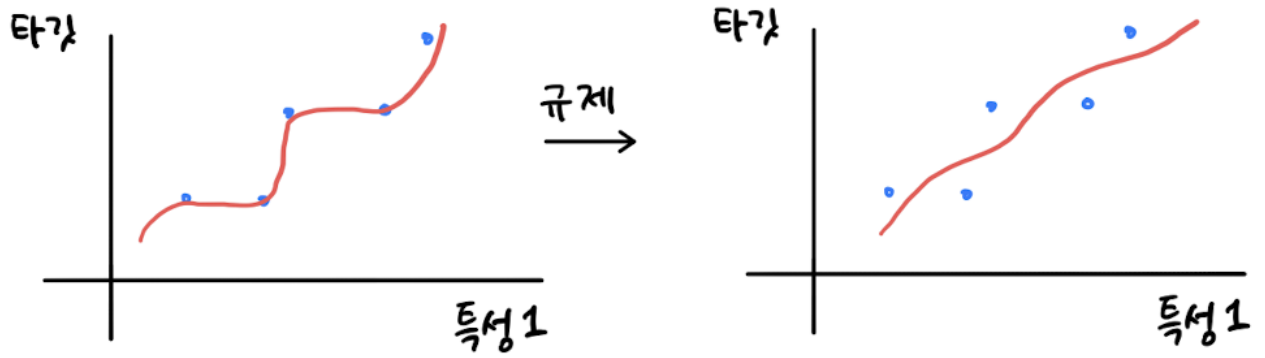
머신러닝 모델이 학습 데이터셋을 너무 과도하게 학습하지 못하도록 훼방하는 것을 말함

- ✓ 즉, 모델이 과대적합되지 않도록 만드는 것임
- ✓ 선형 회귀 모델의 경우, 특성에 곱해지는 계수(또는 기울기)의 크기를 작게 만드는 일임

04. 규제

❖ 규제 이해하기 (2/2)

- 이해를 돕기 위해, 아래와 같이 하나의 특성을 가진 데이터를 학습한 모델을 생각해 보겠음



✓ 왼쪽은 학습 데이터셋을 과도하게 학습하였음
✓ 오른쪽은 기울기를 줄여, 보다 보편적인 패턴을 학습하고 있음

**앞서 55개의 특성으로 훈련한 선형 회귀 모델의 계수를 규제하여,
학습 데이터셋의 점수를 낮추고, 대신 시험 데이터셋의 점수를 높여 보자**

[사진출처] 혼자 공부하는 머신러닝+딥러닝 (출판사: 한빛아카데미)

04. 규제

❖ 표준화 (1/5)

- 규제 적용하기 전에, 특성의 스케일(Scale)에 대해 잠시 생각해 보자
- 우리가 지금 사용하고 있는 특성들은 갖는 값의 범위가 서로 다름
- 이를 두 특성의 스케일이 다르다고 말함
- 어떤 사람이 방의 넓이를 재는데 세로는 cm로, 가로는 inch로 쟀다면, 정사각형인 방도 직사각형처럼 보일 것임

✓ 데이터를 표현하는 기준이 다르면 알고리즘이 올바르게 예측할 수 없음
✓ 제대로 사용하려면 특성값을 일정한 기준으로 맞춰야 함

04. 규제

❖ 표준화 (2/5)

- 특성의 스케일이 서로 다르면, 특성에 곱해지는 계수 값도 차이 나게 됨
- 일반적으로 선형 회귀 모델에 규제를 적용할 때, 계수 값의 크기가 서로 많이 다르면 공정하게 제어되지 않을 것임

규제를 적용하기 전에 먼저 Feature Scaling을 해야 함!

✓ 데이터를 표현하는 기준이 다르면 알고리즘이 올바르게 예측할 수 없음
✓ 제대로 사용하려면 특성값을 일정한 기준으로 맞춰야 함

04. 규제

❖ 표준화 (3/5)

- 가장 널리 사용되는 Feature Scaling 기술 중 하나는, 바로 표준점수(Standard Score, Z-Score)임
- 표준점수는 각 특성값이 평균에서 표준편차의 몇 배만큼 떨어져 있는지를 나타냄
- 이를 통해 실제 특성값의 크기와 상관없이 동일한 조건으로 비교할 수 있음

표준점수와 표준편차

✓ 분산(Variance)은 데이터에서 평균을 뺀 값을 모두 제곱한 다음 평균을 내어 구함
✓ 표준편차(Standard Deviation)은 분산의 제곱근으로 데이터가 분산된 정도를 나타냄
✓ 표준점수는 각 데이터가 원점에서 몇 표준편차만큼 떨어져 있는지를 나타내는 값임

04. 규제

❖ 표준화 (4/5)

- 계산하는 방법은 간단한데, 평균을 빼고 표준편차로 나누어 주면 됨
- 평균과 표준편차를 직접 구해서, 특성을 표준점수로 바꿀 수 있음
- 매우 고맙게도 scikit-learn은 특성을 표준점수로 바꿔주는 StandardScaler 클래스를 제공함
- 우리 수업에서는 scikit-learn의 StandardScaler 클래스 이용하여, 특성을 표준점수로 바꾸어 보자
- StandardScaler 클래스도 변환기의 하나임

04. 규제

❖ 표준화 (5/5)

- 우선 아래의 코드를 따라서 입력해 보자

```
1 from sklearn.preprocessing import StandardScaler
2
3 ss = StandardScaler()
4 ss.fit(x_train_poly)
5
6 x_train_scaled = ss.transform(x_train_poly)
7 x_test_scaled = ss.transform(x_test_poly)
```

✓ 먼저 StandardScaler 클래스의 객체 ss를 초기화한 후, PolynomialFeatures 클래스로 만든 x_train_poly를 사용해 이 객체를 훈련함
✓ 여기에서도 다시 한번 강조하지만, 꼭 학습 데이터셋으로 학습한 변환기를 사용해서 시험 데이터셋까지 변환해야 함
✓ 이제 표준점수로 변환한 x_train_scaled와 x_test_scaled가 준비되었음

✓ 학습 데이터셋에서 학습한 평균과 표준편차는 StandardScaler 클래스 객체의 mean_, scale_ 속성에 저장됨
✓ 특성마다 계산하므로 55개의 평균과 표준편차가 들어 있음

04. 규제

❖ 릿지와 라쏘

선형 회귀 모델에 규제를 추가한 모델을 릿지(Ridge)와 라쏘(Lasso)라고 부름

- ✓ 두 모델은 규제를 가하는 방법이 다름
- ✓ 릿지는 계수를 제곱한 값을 기준으로 규제를 적용함
- ✓ 라쏘는 계수의 절대값을 기준으로 규제를 적용함
- ✓ 두 알고리즘 모두 계수의 크기를 줄이지만, 라쏘는 아예 0으로 만들 수도 있음
- ✓ scikit-learn은 이 두 알고리즘을 모두 제공함

04. 규제

❖ 릿지 회귀 (1/11)

- 릿지와 라쏘 모두 sklearn.linear_model 패키지 안에 있음
- scikit-learn 모델을 사용할 때 편리한 점은 사용하는 방법이 항상 같다는 것임

① 모델 객체 만들기
② fit( ) 메서드를 사용하여 모델 훈련시키기
③ score( ) 메서드를 사용하여 모델 평가하기

04. 규제

❖ 릿지 회귀 (2/11)

- 앞서 준비한 x_train_scaled 데이터로 릿지 모델을 훈련해 보자

```
1 from sklearn.linear_model import Ridge
2
3 ridge = Ridge()
4 ridge.fit(x_train_scaled, y_train)
5 print(ridge.score(x_train_scaled, y_train))
```

실행결과

```
0.9896217956447125
```

선형 회귀에서 거의 완벽에 가까웠던 점수와 비교했을 때, 점수가 조금 낮아졌음

04. 규제

❖ 릿지 회귀 (3/11)

- 이번에는 시험 데이터셋에 대한 점수를 확인해 보자

```
1 print(ridge.score(x_test_scaled, y_test))
```

실행결과

```
0.9788853860988027
```

✓ 시험 데이터셋 점수가 정상으로 돌아왔음
✓ 확실히 많은 특성을 사용했음에도 불구하고 학습 데이터셋에 과대적합되지 않아 시험 데이터셋에서도 좋은 성능을 내고 있음

04. 규제

❖ 릿지 회귀 (4/11)

- 릿지와 라쏘 모델을 사용할 때 규제의 양을 임의로 조절할 수 있음
- 모델 객체를 만들 때 alpha 매개변수로 규제의 강도를 조절함
- **alpha 값이 크면** 규제의 강도가 세지므로 계수 값을 더 줄이고
  조금 더 과소적합되도록 유도함
- **alpha 값이 작으면** 계수를 줄이는 역할이 줄어들고
  선형 회귀 모델과 유사해지므로 과대적합될 가능성이 큼

04. 규제

❖ 릿지 회귀 (5/11)

사람이 직접 지정해야 하는 매개변수

- ✓ alpha 값은 릿지 모델이 학습하는 값이 아니라 사전에 우리가 지정해야 하는 값임
- ✓ 이렇게 머신러닝 모델이 학습할 수 없고 사람이 알려줘야 하는 파라미터를 **하이퍼파라미터(Hyperparameter)**라고 부름
- ✓ scikit-learn과 같은 머신러닝 라이브러리에서 하이퍼파라미터는 클래스와 매서드의 매개변수로 표현됨

04. 규제

❖ 릿지 회귀 (6/11)

- 적절한 alpha 값을 찾는 한 가지 방법은, alpha 값에 대한 R^2 값의 그래프를 그려 보는 것임
- 학습 데이터셋과 시험 데이터셋의 점수가 가장 가까운 지점이 최적의 alpha 값이 됨

04. 규제

❖ 릿지 회귀 (7/11)

- 먼저 맷플롯립을 임포트하고 alpha 값을 바꿀 때마다, score( ) 메서드의 결과를 저장할 리스트를 만듦

```
1 import matplotlib.pyplot as plt
2
3 train_score = []
4 test_score = []
```

04. 규제

❖ 릿지 회귀 (8/11)

- alpha 값을 0.001에서 100까지 10배씩 늘려가며 릿지 회귀 모델을 훈련한 다음, 학습 데이터셋과 시험 데이터셋의 점수를 파이썬 리스트에 저장함

```
alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]

for alpha in alpha_list:
  # 릿지 모델을 만듦
  ridge = Ridge(alpha=alpha)
  # 릿지 모델을 훈련함
  ridge.fit(x_train_scaled, y_train)
  # 학습 데이터셋, 시험 데이터셋에 대한 점수를 저장함
  train_score.append(ridge.score(x_train_scaled, y_train))
  test_score.append(ridge.score(x_test_scaled, y_test))
```

04. 규제

❖ 릿지 회귀 (9/11)

- 그래프를 그려보자
- alpha 값을 0.001부터 10배씩 늘렸기 때문에 이대로 그래프를 그리면, 그래프 왼쪽이 너무 촘촘해 짐
- alpha_list에 있는 6개의 값을 동일한 간격으로 나타내기 위해서, 로그 함수로 바꾸어 지수로 표현하자
- 즉, 0.001은 -3, 0.01은 -2가 되는 식임

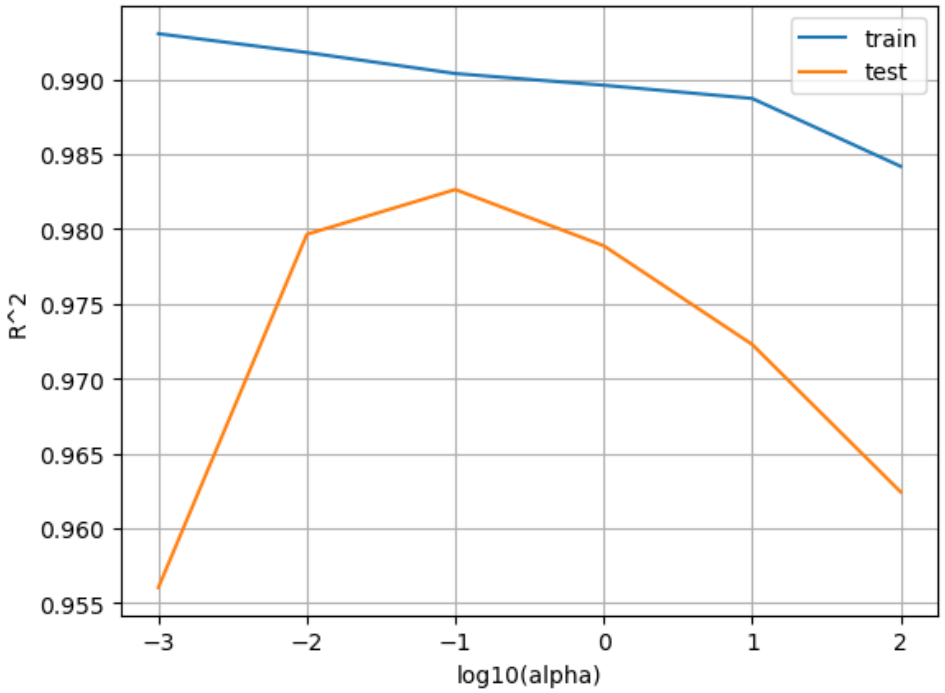
```
1  import numpy as np
2
3  plt.figure()
4  plt.plot(np.log10(alpha_list), train_score, label="train")
5  plt.plot(np.log10(alpha_list), test_score, label="test")
6  plt.xlabel("log10(alpha)")
7  plt.ylabel("R^2")
8  plt.legend()
9  plt.grid(True)
10 plt.show()
```

✓ 넘파이 로그 함수는 np.log( )와 np.log10( )이 있음
✓ np.log( )는 자연 상수 e를 밑으로 하는 자연로그임
✓ np.log10( )은 10을 밑으로 하는 상용로그임

04. 규제

❖ 릿지 회귀 (10/11)

실행결과



적절한 alpha 값은 두 그래프가 가장 가깝고 시험 데이터셋의 점수가 가장 높은 -1, 즉 10^{-1}=0.1임

- ✓ 위 그래프의 왼쪽을 보면 학습 데이터셋과 시험 데이터셋의 점수 차이가 아주 큼
- ✓ 학습 데이터셋에는 잘 맞고 시험 데이터셋에는 형편없는 과대적합의 전형적인 모습임
- ✓ 반대로 오른쪽 편은 학습 데이터셋과 시험 데이터셋의 점수가 모두 낮아지는 과소적합으로 가는 모습을 보임

04. 규제

❖ 릿지 회귀 (11/11)

- alpha 값을 0.1로 하여 최종 모델을 훈련하자

```
1 ridge = Ridge(alpha=0.1)
2 ridge.fit(x_train_scaled, y_train)
3 print(ridge.score(x_train_scaled, y_train))
4 print(ridge.score(x_test_scaled, y_test))
```

실행결과

```
0.990404845594141
0.9826465162736733
```

학습 데이터셋과 시험 데이터셋의 점수가 비슷하게 모두 높고
과대적합과 과소적합 사이에서 균형을 맞추고 있음

그럼 이번에는 라쏘 모델을 훈련해 보자

04. 규제

❖ 라쏘 회귀 (1/7)

- 라쏘 모델을 훈련하는 것은 릿지와 매우 비슷함
- Ridge 클래스를 Lasso 클래스로 바꾸는 것이 전부임

```
1 from sklearn.linear_model import Lasso
2
3 lasso = Lasso()
4 lasso.fit(x_train_scaled, y_train)
5 print(lasso.score(x_train_scaled, y_train))
6 print(lasso.score(x_test_scaled, y_test))
```

실행결과

```
0.9898014198970121
0.9798798667260247
```

시험 데이터셋의 점수가 릿지만큼 좋음

04. 규제

❖ 라쏘 회귀 (2/7)

- 라쏘 모델도 alpha 매개변수로 규제의 강도를 조절할 수 있음
- 앞에서와 같이 alpha 값을 바꾸어 가며 학습 데이터셋과 시험 데이터셋에 대한 점수를 계산하자

```
train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]

for alpha in alpha_list:
  # 라쏘 모델을 만듦
  lasso = Lasso(alpha=alpha)
  # 라쏘 모델을 훈련함
  lasso.fit(x_train_scaled, y_train)
  # 학습 데이터셋, 시험 데이터셋에 대한 점수를 저장함
  train_score.append(lasso.score(x_train_scaled, y_train))
  test_score.append(lasso.score(x_test_scaled, y_test))
```

04. 규제

❖ 라쏘 회귀 (3/7)

경고(Warning)가 뜨는데, 정상인가요?

- ✓ 라쏘 모델을 훈련할 때 **ConvergenceWaring이란 경고가 발생**할 수 있음
- ✓ scikit-learn의 라쏘 모델은 최적의 계수를 찾기 위해 반복적인 계산을 수행하는데, 지정한 반복 횟수가 부족할 때 이런 경고가 발생함
- ✓ 이 반복 횟수를 충분히 늘리려면, max_iter 매개변수의 값에 10,000을 지정하면 됨
- ✓ 필요하면 더 늘릴 수 있지만, 이 문제에서는 큰 영향을 끼치지 않음

수렴

목적함수 : Minimize MSE or MAE

```
\usr\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647:
ConvergenceWarning: Objective did not converge. You might want to increase the number
of iterations, check the scale of the features or consider increasing regularisation.
1.872e+04, tolerance: 5.183e+02 model = cd_fast.enet_coordinate_descent(
```

최적화(optimization)

04. 규제

❖ 라쏘 회귀 (4/7)

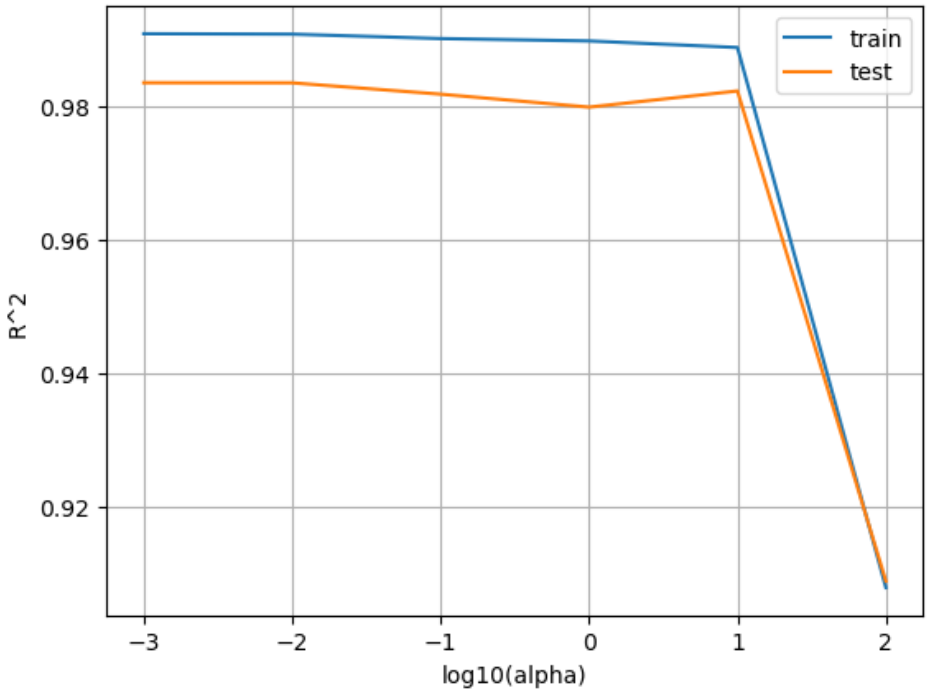
- 그다음 train_score와 test_score 리스트를 사용해 그래프를 그려 보자
- 이 그래프도 x축은 로그 스케일로 바꾸자

```
plt.figure()
plt.plot(np.log10(alpha_list), train_score, label="train")
plt.plot(np.log10(alpha_list), test_score, label="test")
plt.xlabel("log10(alpha)")
plt.ylabel("R^2")
plt.legend()
plt.grid(True)
plt.show()
```

04. 규제

❖ 라쏘 회귀 (5/7)

실행결과



라쏘 모델에서 최적의 alpha 값은 1, 즉 10^1=10임

✓ 위 그래프도 왼쪽은 과대적합을 보여줌
✓ 오른쪽으로 갈수록 학습 데이터셋과 시험 데이터셋의 점수가 좁혀지고 있음
✓ 가장 오른쪽은 아주 크게 점수가 떨어짐
✓ 이 지점은 분명 과소적합되는 모델일 것임

04. 규제

❖ 라쏘 회귀 (6/7)

- alpha 값을 10으로 설정하고, 다시 모델을 훈련하자

```
1 lasso = Lasso(alpha=10)
2 lasso.fit(x_train_scaled, y_train)
3 print(lasso.score(x_train_scaled, y_train))
4 print(lasso.score(x_test_scaled, y_test))
```

실행결과

```
0.988820885788649
0.9823020708550176
```

✓ 모델이 잘 훈련되었음
✓ 특성을 많이 사용했지만, 릿지와 마찬가지로 라쏘 모델이 과대적합을 잘 억제하고 시험 데이터셋의 성능을 높였음

04. 규제

❖ 라쏘 회귀 (7/7)

- 앞에서 라쏘 모델은 계수 값을 아예 0으로 만들 수 있다고 했음
- 라쏘 모델의 계수는 coef_ 속성에 저장되어 있음
- 이 중에 0인 것을 헤아려 보자

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

실행결과

```
40
```

✓ np.sum( ) 함수는 배열을 모두 더한 값을 반환함
✓ 넘파이 배열에 비교 연산자를 사용했을 때, 각 원소는 True 또는 False가 됨
✓ np.sum( ) 함수는 True를 1로, False를 0으로 인식하여 덧셈을 할 수 있기 때문에 마치 비교 연산자에 맞는 원소 개수를 헤아리는 효과를 냄
✓ 40개나 되는 특성의 계수가 모두 0이 되었음
✓ 55개의 특성을 모델에 주입했지만, 라쏘 모델이 사용한 특성은 15개 밖에 되지 않음

이런 특징 때문에 라쏘 모델을 유용한 특성을 골라내는 용도로도 사용할 수 있음

05. 모델의 과대적합을 제어하기

01. 다중 회귀
02. 사이킷런의 변환기
03. 다중 회귀 모델 훈련하기
04. 규제
06. 마무리

05. 모델의 과대적합을 제어하기

❖ 전체 과정 요약 (1/2)

- 우리는 선형 회귀 알고리즘을 사용해 농어의 무게를 예측하는 모델을 훈련시켰지만,
  학습 데이터셋에 과소적합되는 문제가 발생했음
- 이를 위해 농어의 길이뿐만 아니라 높이와 두께도 사용하여 다중 회귀 모델을 훈련시켰음
- 또한 다항 특성을 많이 추가하여 학습 데이터셋에 거의 완벽에 가까운 점수를 얻는 모델을 훈련했음
- 특성을 많이 추가하면 선형 회귀는 매우 강력한 성능을 냄
- 하지만 특성이 너무 많으면 과대적합 문제가 발생함

05. 모델의 과대적합을 제어하기

❖ 전체 과정 요약 (2/2)

- 이를 위해 릿지 회귀와 라쏘 회귀에 대해 알아보았음
- scikit-learn을 사용해 다중 회귀 모델과, 릿지, 라쏘 모델을 훈련시켰음
- 또 릿지와 라쏘 모델의 규제 양을 조절하기 위한 최적의 alpha 값을 찾는 방법을 알아보았음

05. 모델의 과대적합을 제어하기

❖ 전체 소스 코드 (1/5)

```
# 01. 다중 회귀
import pandas as pd

df = pd.read_csv("Fish.csv")

print(type(df))
df

df = df[df["Species"] == "Perch"]
df

df_features = df[["Length2", "Height", "Width"]]
df_label = df[["Weight"]]

perch_full = df_features.to_numpy()
print(perch_full)

perch_weight = df_label.to_numpy()
print(perch_weight)

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =
train_test_split(perch_full, perch_weight,
                 random_state=42)
```

```
# 02. 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([[2, 3]])
print(poly.transform([[2, 3]]))

poly = PolynomialFeatures(include_bias=False)
poly.fit(x_train)
x_train_poly = poly.transform(x_train)
print(x_train_poly.shape)

poly.get_feature_names_out()

x_test_poly = poly.transform(x_test)
```

05. 모델의 과대적합을 제어하기

❖ 전체 소스 코드 (2/5)

```
# 03. 다중 회귀 모델 훈련하기
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x_train_poly, y_train)

print(lr.score(x_train_poly, y_train))

print(lr.score(x_test_poly, y_test))

poly = PolynomialFeatures(degree=5, include_bias=False)
poly.fit(x_train)
x_train_poly = poly.transform(x_train)
x_test_poly = poly.transform(x_test)
print(x_train_poly.shape)

lr.fit(x_train_poly, y_train)
print(lr.score(x_train_poly, y_train))

print(lr.score(x_test_poly, y_test))
```

```
# 04. 규제
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(x_train_poly)

x_train_scaled = ss.transform(x_train_poly)
x_test_scaled = ss.transform(x_test_poly)

## 릿지 회귀
from sklearn.linear_model import Ridge

ridge = Ridge()
ridge.fit(x_train_scaled, y_train)
print(ridge.score(x_train_scaled, y_train))

print(ridge.score(x_test_scaled, y_test))
```

05. 모델의 과대적합을 제어하기

❖ 전체 소스 코드 (3/5)

```
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 만듦
  ridge = Ridge(alpha=alpha)
  # 릿지 모델을 훈련함
  ridge.fit(x_train_scaled, y_train)
  # 학습 데이터셋, 시험 데이터셋에 대한 점수를 저장함
  train_score.append(ridge.score(x_train_scaled,
                                 y_train))
  test_score.append(ridge.score(x_test_scaled,
                                y_test))
```

```
import numpy as np

plt.figure()
plt.plot(np.log10(alpha_list), train_score,
         label="train")
plt.plot(np.log10(alpha_list), test_score,
         label="test")
plt.xlabel("alpha")
plt.ylabel("R^2")
plt.legend()
plt.grid(True)
plt.show()

ridge = Ridge(alpha=0.1)
ridge.fit(x_train_scaled, y_train)
print(ridge.score(x_train_scaled, y_train))
print(ridge.score(x_test_scaled, y_test))
```

05. 모델의 과대적합을 제어하기

❖ 전체 소스 코드 (4/5)

```
## 라쏘 회귀
from sklearn.linear_model import Lasso

lasso = Lasso()
lasso.fit(x_train_scaled, y_train)
print(lasso.score(x_train_scaled, y_train))
print(lasso.score(x_test_scaled, y_test))

train_score = []
test_score = []

alpha_list = [0.001, 0.01, 0.1, 1, 10, 100]
```

```
for alpha in alpha_list:
  # 라쏘 모델을 만듦
  lasso = Lasso(alpha = alpha, max_iter=10000)
  # 라쏘 모델을 훈련함
  lasso.fit(x_train_scaled, y_train)
  # 학습 데이터셋, 시험 데이터셋에 대한 점수를 저장함
  train_score.append(lasso.score(x_train_scaled,
                                 y_train))
  test_score.append(lasso.score(x_test_scaled,
                                y_test))
plt.figure()
plt.plot(np.log10(alpha_list), train_score,
         label="train")
plt.plot(np.log10(alpha_list), test_score,
         label="test")
plt.xlabel("alpha")
plt.ylabel("R^2")
plt.legend()
plt.grid(True)
plt.show()
```

05. 모델의 과대적합을 제어하기

❖ 전체 소스 코드 (5/5)

```
161  lasso = Lasso(alpha=10)
162  lasso.fit(x_train_scaled, y_train)
163  print(lasso.score(x_train_scaled, y_train))
164  print(lasso.score(x_test_scaled, y_test))
165
166  print(np.sum(lasso.coef_ == 0))
```

06. 마무리

01. 다중 회귀
02. 사이킷런의 변환기
03. 다중 회귀 모델 훈련하기
04. 규제
05. 모델의 과대적합을 제어하기

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (1/3)

다중 회귀(Multiple Regression)

- ✓ 여러 개의 특성을 사용하는 회귀 모델임
- ✓ 특성이 많으면 선형 모델은 강력한 성능을 발휘함

특성 공학(Feature Engineering)

- ✓ 주어진 특성을 조합하여 새로운 특성을 만드는 일련의 작업 과정임

표준점수(Standard Score, Z-Score)

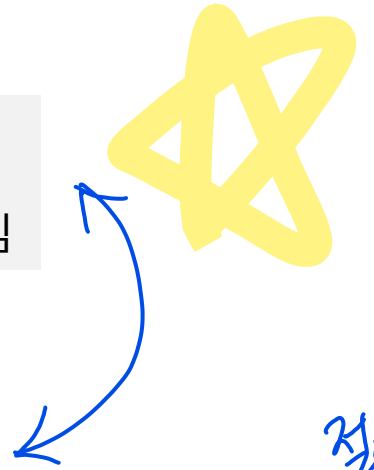
- ✓ 학습 데이터셋의 스케일(Scale)을 바꾸는 대표적인 방법 중 하나임
- ✓ 표준점수를 얻으려면 특성의 평균을 빼고 표준편차로 나눔
- ✓ 반드시 학습 데이터셋의 평균과 표준편차로 시험 데이터셋을 바꿔야 함

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (2/3)

릿지(Ridge)

- ✓ 규제가 있는 선형 회귀 모델 중 하나임
- ✓ 선형 모델의 계수를 작게 만들어 과대적합을 완화시킴
- ✓ 릿지는 비교적 효과가 좋아 널리 사용하는 규제 방법임



라쏘(Lasso)

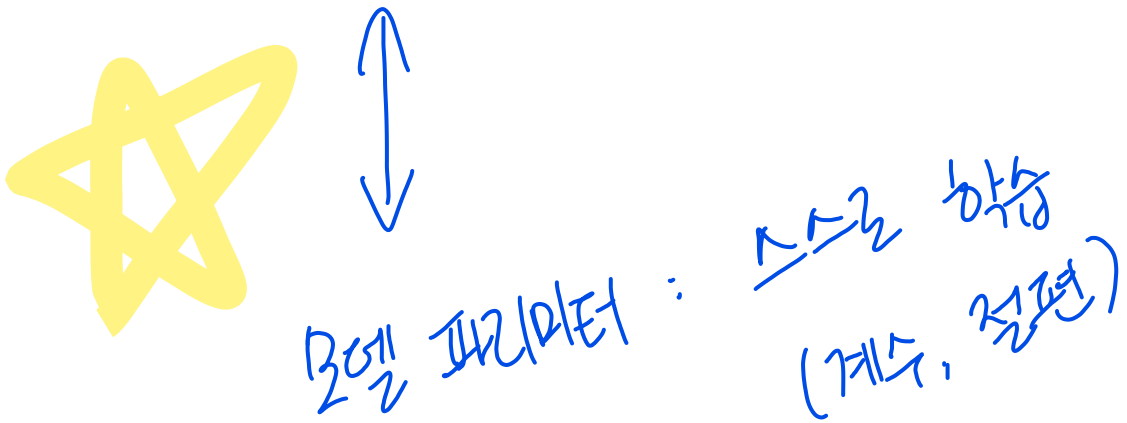
- ✓ 또 다른 규제가 있는 선형 회귀 모델임
- ✓ 릿지와 달리 계수 값을 아예 0으로 만들 수도 있음

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (3/3)

하이퍼파라미터(Hyperparameter)

- ✓ 머신러닝 알고리즘이 학습하지 않는 파라미터임
- ✓ 이런 파라미터는 사람이 사전에 지정해야 함
- ✓ 릿지와 라쏘에서는 규제 강도 alpha가 하이퍼파라미터임



06. 마무리

❖ 핵심 패키지와 함수: pandas

read_csv( )

✓ CSV 파일을 로컬 컴퓨터나 인터넷에서 읽어 pandas DataFrame으로 변환하는 함수임
✓ 이 함수는 매우 많은 매개변수를 제공함
✓ 그중에 자주 사용하는 매개변수는 다음과 같음

No.	매개변수	설명
1	sep	CSV 파일의 구분자를 지정하며, 기본값은 콤마(,)임
2	header	DataFrame의 열 이름으로 사용할 CSV 파일의 행 번호를 지정함 (기본적으로 첫 번째 행을 열 이름으로 사용함)
3	skiprows	파일에서 읽기 전에 건너뛸 행의 개수를 지정함
4	nrows	파일에서 읽을 행의 개수를 지정함

06. 마무리

❖ 핵심 패키지와 함수: scikit-learn (1/3)

PolynomialFeatures

✓ 주어진 특성을 조합하여 새로운 특성을 만듦

No.	매개변수	설명
1	degree	최고 차수를 지정하며, 기본값은 2임
2	interaction_only	True이면 거듭제곱 항은 제외되고, 특상 간의 곱셈 항만 추가됨 (기본값은 False)
3	include_bias	False이면, 절편을 위한 특성을 추가하지 않음 (기본값은 True)

❖ 핵심 패키지와 함수: scikit-learn (2/3)

Ridge

✓ 규제가 있는 회귀 알고리즘인 릿지 회귀 모델을 훈련함

No.	매개변수	설명
1	alpha	✓ 규제의 강도를 조절함 ✓ 값이 클수록 규제가 세지며, 기본값은 1임
2	solver	✓ 최적의 모델을 찾기 위한 방법을 지정할 수 있음 ✓ 기본값은 'auto'이며 데이터에 따라 자동으로 선택됨 ✓ scikit-learn 0.17 버전에 추가된 'sag'는 확률적 평균 경사 하강법 알고리즘으로, 특성과 샘플 수가 많을 때에 성능이 빠르고 좋음 ✓ scikit-learn 0.19 버전에는 'sag'의 개선 버전인 'saga'가 추가됨
3	random_state	solver가 'sag'나 'saga'일 때 넘파이 난수 시드값을 지정할 수 있음

❖ 핵심 패키지와 함수: scikit-learn (3/3)

Lasso

✓ 규제가 있는 회귀 알고리즘인 라쏘 회귀 모델을 훈련함
✓ 이 클래스는 최적의 모델을 찾기 위해 좌표축을 따라 최적화를 수행해가는 좌표 하강법(Coordinate Descent)을 사용함

No.	매개변수	설명
1	alpha	✓ 규제의 강도를 조절함 ✓ 값이 클수록 규제가 세지며, 기본값은 1임
2	max_iter	알고리즘의 수행 반복 횟수를 지정함(기본값은 1000)

06. 마무리

❖ 확인 문제 1.

a, b, c 특성으로 이루어진 학습 데이터셋을 PolynomialFeautres(degree=3)로 변환했습니다. 다음 중 변환된 데이터에 포함되지 않는 특성은 무엇인가요?

① 1

② a^1

③ $a^1 \times b^1$

④ $a \times b^3$

06. 마무리

❖ 확인 문제 2.

다음 중 특성을 표준화하는 scikit-learn 변환기 클래스는 무엇인가요?

① Ridge
② Lasso
③ StandardScaler
④ LinearRegression

06. 마무리

❖ 확인 문제 3.

다음 중 과대적합과 과소적합을 올바르게 표현하지 못한 것은 무엇인가요?

① 과대적합인 모델은 학습 데이터셋의 점수가 높습니다.
② 과대적합인 모델은 시험 데이터셋의 점수도 높습니다.
③ 과소적합인 모델은 학습 데이터셋의 점수가 낮습니다.
④ 과소적합인 모델은 시험 데이터셋의 점수도 낮습니다.

끝맺음

- 01. 다중 회귀
- 02. 사이킷런의 변환기
- 03. 다중 회귀 모델 훈련하기
- 04. 규제
- 05. 모델의 과대적합을 제어하기
- 06. 마무리

THANK YOU!
Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: bkwon@dongduk.ac.kr