

Introduction to Machine Learning with Python

4. Representing Data and Engineering Features

Honedae Machine Learning Study Epoch #2

Contacts

Haesun Park

Email : haesunrpark@gmail.com

Meetup: <https://www.meetup.com/Hongdae-Machine-Learning-Study/>

Facebook : <https://facebook.com/haesunrpark>

Blog : <https://tensorflow.blog>

Book

파이썬 라이브러리를 활용한 머신러닝, 박해선.

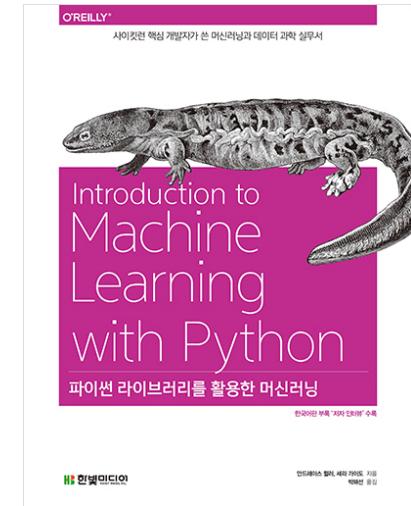
(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

Github:

https://github.com/rickiepark/introduction_to_ml_with_python/



데이터 표현과 특성 공학

연속형 vs 범주형

출력값이 연속형 → 회귀 , 출력값이 범주형 → 분류

입력이 연속형 특성 continuous feature : 실수 데이터, ex) 0.13493, 100.0

입력이 범주형(이산형) 특성 categorical feature : 연속적이지 않은 숫자나 속성,
ex) 픽셀 강도, 브랜드, 쇼핑카테고리

범주형 특성의 사이에는 중간값이 없고 순서가 없습니다. ex) 책과 옷

특성 공학 feature engineering : 애플리케이션에 가장 적합한 데이터 표현을 찾는 것

알맞은 데이터 표현 >>> 매개변수 탐색 ex) inch와 cm의 스케일, 다항식 특성 등

범주형 변수

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[p] \times x[p] + b > 0$$

?

범주형 특성

이진 출력: 분류문제
 $\leq 50, > 50k$

age	workclass	education	gender	hours-per-week	occupation	income
0 39	State-gov	Bachelors	Male	40	Adm-clerical	$\leq 50K$
1 50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	$\leq 50K$
2 38	Private	HS-grad	Male	40	Handlers-cleaners	$\leq 50K$
3 53	Private	11th	Male	40	Handlers-cleaners	$\leq 50K$
4 28	Private	Bachelors	Female	40	Prof-specialty	$\leq 50K$

연속형 특성

1994년 인구조사 데이터베이스: 미국 성인의 소득 데이터셋(Adult Data Set)

원-핫-인코딩 one-hot-encoding

범주형 변수를 0 또는 1의 값을 가진 여러개의 새로운 특성으로 바꿈(이진 특성)

원-아웃-오브-엔 인코딩 one-out-of-N encoding, 혹은 가변수 dummy variable이라고도 함

workclass 특성이 4개의 새로운 특성으로 바뀜(workclass는 더 이상 사용하지 않음)

workclass	Government Employee	Private Employee	Self Employed	Self Employed Incorporated
Government Employee	1	0	0	0
Private Employee	0	1	0	0
Self Employed	0	0	1	0
Self Employed Incorporated	0	0	0	1

* 통계에서의 더미 인코딩은 마지막 값을 모든 변수가 0인 것으로 대신함(열 랭크 부족 현상 때문)

pandas.read_csv

헤더가 없고, 행 인덱스 없음

```
53, Private, 234721, 11th, 7, Married-civ-spouse, Handlers-cleaners, Husband, Black, Male, 0, 0, 40, United-States, <=50K  
28, Private, 338409, Bachelors, 13, Married-civ-spouse, Prof-specialty, Wife, Black, Female, 0, 0, 40, Cuba, <=50K  
37, Private, 284582, Masters, 14, Married-civ-spouse, Exec-managerial, Wife, White, Female, 0, 0, 40, United-States, <=50K  
49, Private, 160187, 9th, 5, Married-spouse-absent, Other-service, Not-in-family, Black, Female, 0, 0, 16, Jamaica, <=50K  
52, Self-emp-not-inc, 209642, HS-grad, 9, Married-civ-spouse, Exec-managerial, Husband, White, Male, 0, 0, 45, United-States, >50K  
31, Private, 45781, Masters, 14, Never-married, Prof-specialty, Not-in-family, White, Female, 14084, 0, 50, United-States, >50K
```

```
import pandas as pd  
  
data = pd.read_csv(  
    os.path.join(mglearn.datasets.DATA_PATH, "adult.data"), header=None, index_col=False,  
    names=['age', 'workclass', 'fnlwgt', 'education', 'education-num',  
           'marital-status', 'occupation', 'relationship', 'race', 'gender',  
           'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',  
           'income'])  
  
# 예제를 위해 몇개의 열만 선택합니다  
data = data[['age', 'workclass', 'education', 'gender', 'hours-per-week',  
            'occupation', 'income']]
```

DataFrame 객체
(like Excel sheet)

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K

pandas.get_dummies

```
In [4]: print(data.gender.value_counts())
```

```
Male      21790  
Female    10771  
Name: gender, dtype: int64
```

```
data_dummies = pd.get_dummies(data)
```

age	hours-per-week	workclass_?	workclass_Federal-gov	workclass_Local-gov	...	occupation_Tech-support	occupation_Transport-moving	income_<=50K	income_>50K	
0	39	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
1	50	13	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
2	38	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
3	53	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0
4	28	40	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0

	age	workclass	education
0	39	State-gov	Bachelors
1	50	Self-emp-not-inc	Bachelors
2	38	Private	HS-grad

범주형 데이터 자동 변환

로지스틱 회귀 적용

pd.read_csv() → pd.get_dummies() → DataFrame.values → NumPy

DataFrame 객체

```
In [7]: features = data_dummies.loc[:, 'age':'occupation_Transport-moving']
# NumPy 배열 출력
X = features.values
y = data_dummies['income_>50K'].values
print("X.shape: {} y.shape: {}".format(X.shape, y.shape))
```

X.shape: (32561, 44) y.shape: (32561,)

occupation_Transport-moving까지 포함됨

```
In [8]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print("테스트 점수: {:.2f}".format(logreg.score(X_test, y_test)))
```

테스트 점수: 0.81

훈련세트 나누기 전에 먼저 get_dummies 적용

원-핫-인코딩 변환시 주의 사항

훈련 세트와 테스트 세트로 나누기 전에 범주형 데이터를 변환합니다.

만약에 훈련 세트와 테스트 세트로 나눈 후에 `get_dummies()` 함수를 사용하면 훈련 세트와 테스트 세트에 각각 다른 인코딩이 적용됩니다.

훈련 세트에만 “Private Employee”가 있다면, 훈련 세트와 테스트 세트의 원-핫-인코딩으로 변환된 특성의 개수가 맞지 않을 것입니다.

훈련 세트에만 “Private Employee”가 있고 테스트 세트에만 “Self Employed”가 있다면, 동일한 원-핫-인코딩이 다른 의미로 사용될 수 있습니다.

반드시 원-핫-인코딩으로 변환한 후에 훈련 데이터와 테스트 데이터로 나눕니다.

숫자로 된 범주형 특성

숫자로 되어 있다고 무조건 연속형 특성은 아닙니다.

ex) workclass를 객관식으로 골랐다면 0, 1, 2, 3 와 같은 숫자로 저장될 수 있습니다.

연속형인지 범주형인지는 특성의 의미를 알아야 판단할 수 있는 경우가 많습니다.

ex) 별 다섯개의 평점 데이터, 영화 관람 등급(범주형이지만 순서가 있음)

pandas의 get_dummies 또는 scikit-learn의 OneHotEncoder를 사용할 수 있습니다.

OneHotEncoder는 숫자로된 범주형 변수에만 사용 가능합니다.

(scikit-learn 0.20 버전에서 문자열로 된 범주형을 원-핫-인코딩으로 변환할 수 있는 CategoricalEncoder 클래스가 추가될 예정입니다)

get_dummies()로 숫자로 된 범주형 특성의 변환

```
In [10]: display(pd.get_dummies(demo_df))
```

	범주형 특성	숫자 특성		숫자 특성	범주형 특성_상자	범주형 특성_양말	범주형 특성_여우
0	양말	0		0	0	1	0
1	여우	1		1	0	0	1
2	양말	2		2	0	1	0
3	상자	1		3	1	0	0

```
In [11]: demo_df['숫자 특성'] = demo_df['숫자 특성'].astype(str)  
display(pd.get_dummies(demo_df, columns=['숫자 특성', '범주형 특성']))
```

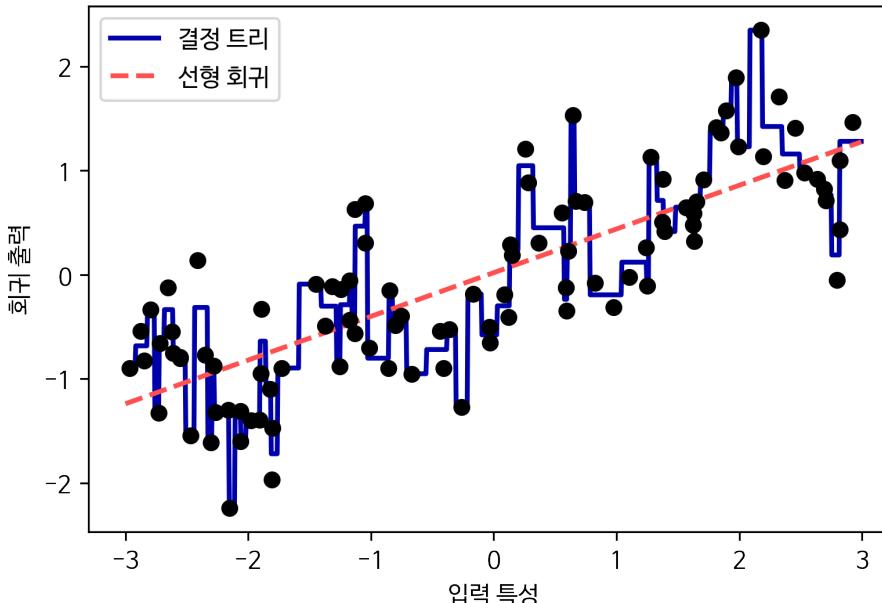
	숫자 특성_0	숫자 특성_1	숫자 특성_2	범주형 특성_상자	범주형 특성_양말	범주형 특성_여우
0	1	0	0	0	1	0
1	0	1	0	0	0	1
2	0	0	1	0	1	0
3	0	1	0	1	0	0

구간 분할

wave + 선형 회귀, 결정트리 회귀

```
reg = DecisionTreeRegressor(min_samples_split=3).fit(X, y)
plt.plot(line, reg.predict(line), label="결정 트리")
```

```
reg = LinearRegression().fit(X, y)
plt.plot(line, reg.predict(line), '--', label="선형 회귀")
```



규제를 조금 추가해
모든 데이터 포인트를 지나지
않도록 했습니다.

구간 분할 binning

연속형 특성 하나를 구간을 나누어 여러개의 범주형 특성으로 만듭니다.

```
In [13]: bins = np.linspace(-3, 3, 11)
print("bins: {}".format(bins))
```

[0] : $x < -3$

bins: [-3. -2.4 -1.8 -1.2 -0.6 0. 0.6 1.2 1.8 2.4 3.]

```
In [14]: which_bin = np.digitize(X, bins=bins)
print("\n데이터 포인트:\n", X[:5])
print("\n데이터 포인트의 소속 구간:\n", which_bin[:5])
```

연속형

데이터 포인트:
[[-0.753]
[2.704]
[1.392]
[0.592]
[-2.064]]

[4] : $-1.2 \leq x < -0.6$

범주형

데이터 포인트의 소속 구간:
[[4]
[10]
[8]
[6]
[2]]

[11] : $3 \leq x$

[10] : $2.4 \leq x < 3$

OneHotEncoder

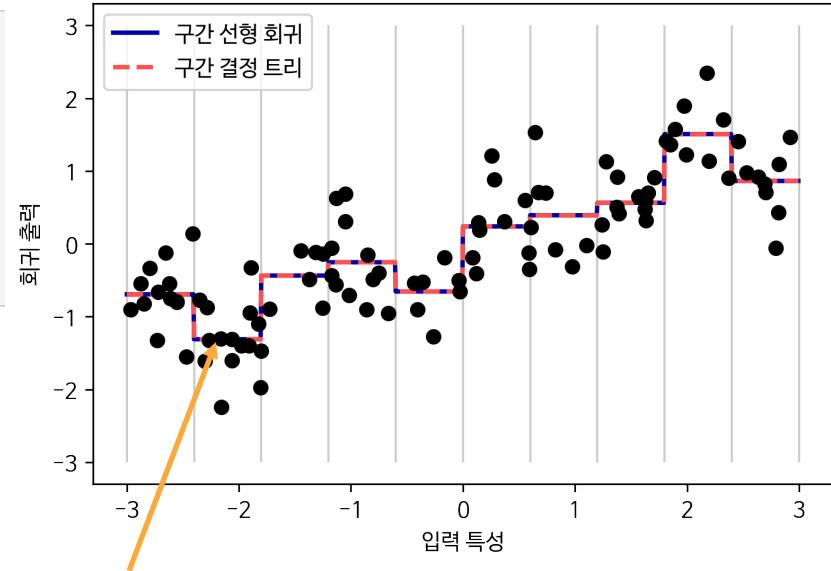
숫자로 된 범주형 특성을 원-핫-인코딩으로 변환합니다.

```
In [15]: from sklearn.preprocessing import OneHotEncoder  
# 변환을 위해 OneHotEncoder를 사용합니다  
encoder = OneHotEncoder(sparse=False)  
# encoder.fit은 which_bin에 나타난 유일한 값을 찾습니다  
encoder.fit(which_bin)  
# 원-핫-인코딩으로 변환합니다  
X_binned = encoder.transform(which_bin)  
print(X_binned[:5])
```

```
[[ 4]  [[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]  
[10]  [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]  
[ 8]  [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]  
[ 6]  [ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]  
[ 2]]  [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

일반 numpy 배열을 반환합니다

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[p] \times x[p] + b$$



선형 모델은 상수 특성으로 인해 유연해졌으나
결정트리는 오히려 더 나빠졌음(결정트리는
스스로 좋은 구간을 학습합니다)

상호작용과 다행식

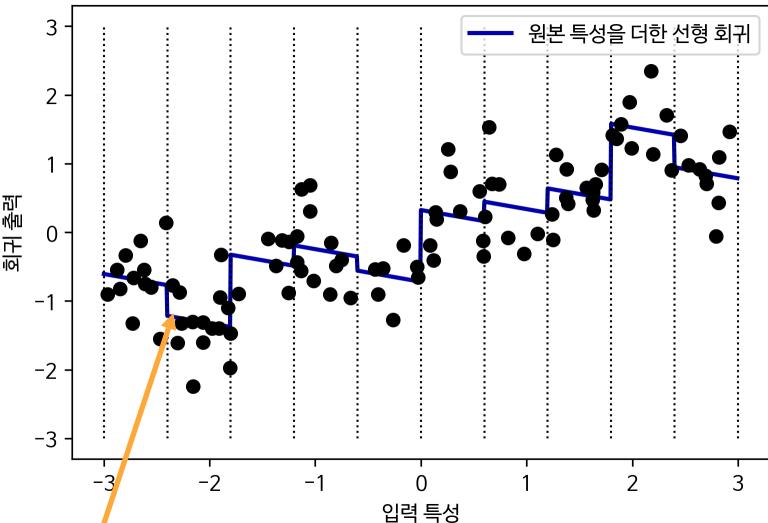
구간 분할 + 원본 데이터

```
In [18]: X_combined = np.hstack([X, X_binned])
print(X_combined.shape)
```

(100, 11)

```
[-0.753,  0. ,  0. ,  ...,  0. ,  0. ,  0. ,  0. ]
[ 2.704,  0. ,  0. ,  ...,  0. ,  0. ,  1. ,  0. ]
[ 1.392,  0. ,  0. ,  ...,  1. ,  0. ,  0. ,  0. ]
...
[-0.435,  0. ,  0. ,  ...,  0. ,  0. ,  0. ,  0. ]
[-2.847,  1. ,  0. ,  ...,  0. ,  0. ,  0. ,  0. ]
[-2.353,  0. ,  1. ,  ...,  0. ,  0. ,  0. ,  0. ]
```

```
In [37]: reg = LinearRegression().fit(X_combined, y)
```



각 구간의 기울기를 하나의 특성으로 학습합니다.

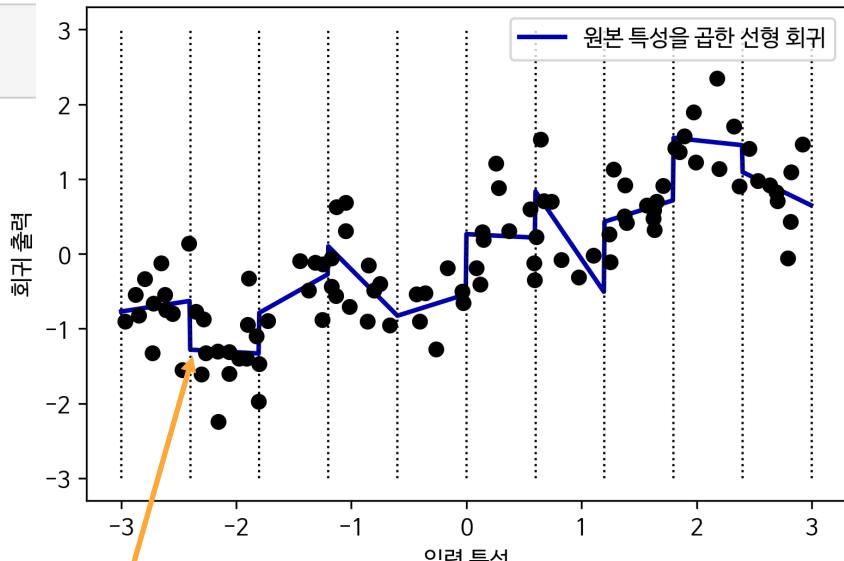
구간 분할 * 원본 데이터

```
In [39]: X_product = np.hstack([X_binned, X * X_binned])
print(X_product.shape)
```

```
(100, 20)
```

```
[ 0. ,  0. ,  0. ,  ..., -0. , -0. , -0. ]
[ 0. ,  0. ,  0. ,  ...,  0. ,  0. ,  2.704]
[ 0. ,  0. ,  0. ,  ...,  1.392,  0. ,  0. ]
...
[ 0. ,  0. ,  0. ,  ..., -0. , -0. , -0. ]
[ 1. ,  0. ,  0. ,  ..., -0. , -0. , -0. ]
[ 0. ,  1. ,  0. ,  ..., -0. , -0. , -0. ]
```

```
reg = LinearRegression().fit(X_product, y)
```



각 구간의 기울기를 10개의 특성에서 따로 학습합니다.

PolynomialFeatures

원본 특성에 상호작용이나 제곱항을 추가합니다(가령 $x[0]**2$, $x[0]*x[1]$ 등)

```
In [22]: from sklearn.preprocessing import PolynomialFeatures  
  
# x ** 10까지 고차항을 추가합니다  
# 기본값인 "include_bias=True"는 절편에 해당하는 1인 특성을 추가합니다  
poly = PolynomialFeatures(degree=10, include_bias=False)  
poly.fit(X)  
X_poly = poly.transform(X)
```



```
In [23]: print("X_poly.shape: {}".format(X_poly.shape))
```

X_poly.shape: (100, 10)

[-0.753]	[1	-0.753	0.567	-0.427	0.321	-0.242	0.182
	-0.137	0.103	-0.078	0.058]			
[2.704]	[2.704	7.313	19.777	53.482	144.632	391.125	
	1057.714	2860.36	7735.232	20918.278]			
[1.392]	[1.392	1.938	2.697	3.754	5.226	7.274	
	10.125	14.094	19.618	27.307]			
[0.592]	[0.592	0.35	0.207	0.123	0.073	0.043	
	0.025	0.015	0.009	0.005]			
[-2.064]	[-2.064	4.26	-8.791	18.144	-37.448	77.289	
	-159.516	329.222	-679.478	1402.367]]			

$x^{**1} \sim x^{**10}$

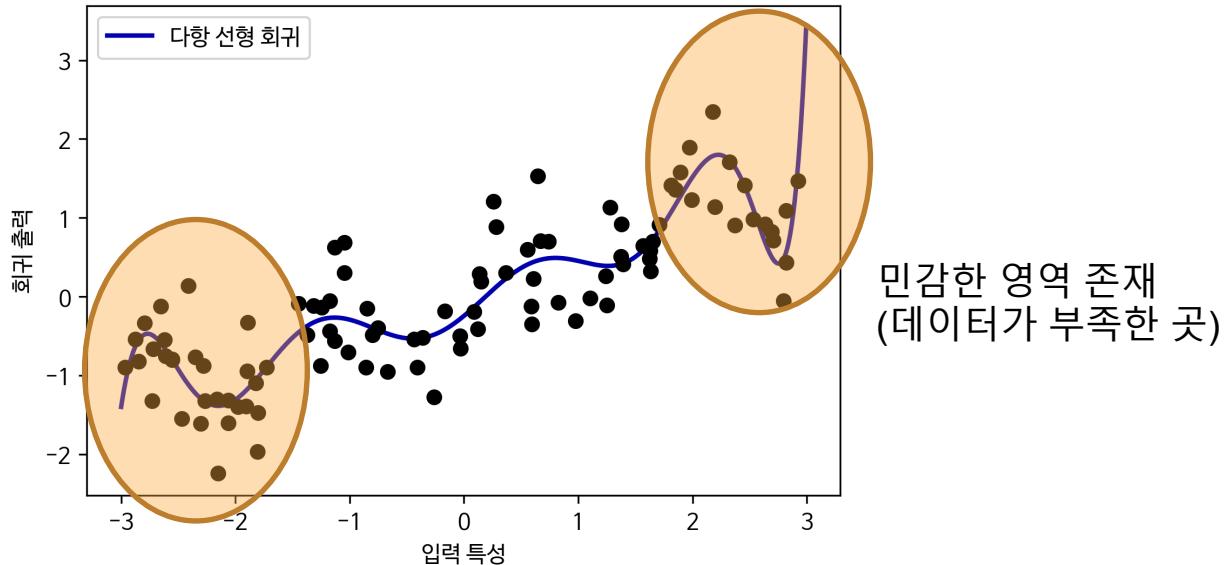
다항 회귀 Polynomial Regression

```
In [25]: print("항 이름:\n{}".format(poly.get_feature_names()))
```

항 이름:

['x0', 'x0^2', 'x0^3', 'x0^4', 'x0^5', 'x0^6', 'x0^7', 'x0^8', 'x0^9', 'x0^10']

```
In [26]: reg = LinearRegression().fit(X_poly, y)
```

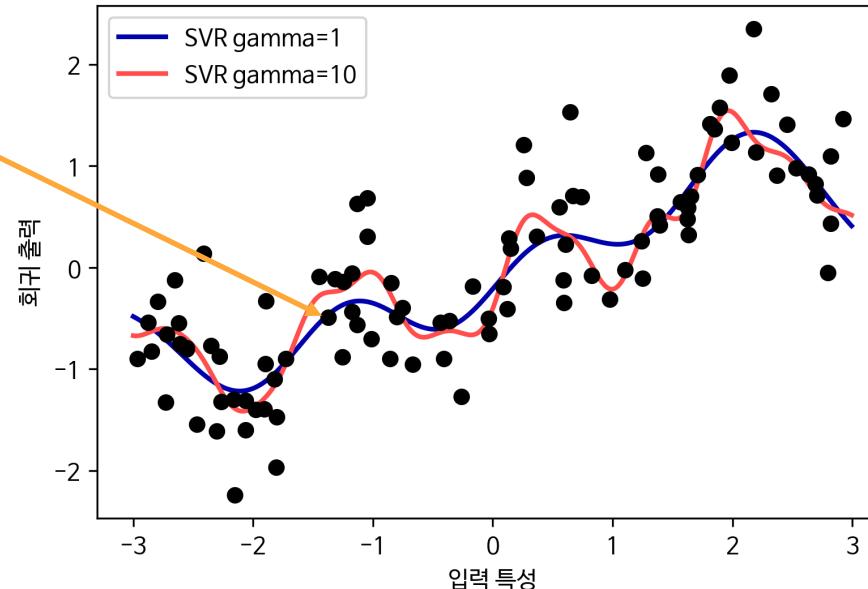


SVR과 비교

```
In [27]: from sklearn.svm import SVR

for gamma in [1, 10]:
    svr = SVR(gamma=gamma).fit(X, y)
    plt.plot(line, svr.predict(line), label='SVR gamma={}'.format(gamma))
```

복잡도가 높은 모델로는
특성을 변환하지 않고도
고차원의 예측을 만듭니다.
(가우시안 커널)



boston dataset + PolynomialFeatures

```
In [29]: poly = PolynomialFeatures(degree=2).fit(X_train_scaled)
X_train_poly = poly.transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_poly.shape: {}".format(X_train_poly.shape))
```

X_train.shape: (379, 13)
X_train_poly.shape: (379, 105)

MinMaxScaler로 0~1 사이로 조정

interaction_only=True 옵션이 추가되면 거듭제곱은 모두 제외됩니다.

중복을 포함한 조합

$$\binom{13}{2} = \frac{14!}{2! 12!} = 91$$

```
In [30]: print("다항 특성 이름: \n{}".format(poly.get_feature_names()))
```

다항 특성 이름:

['1', 'x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x0^2', 'x0 x1', 'x0 x2', 'x0 x3', 'x0 x4', 'x0 x5', 'x0 x6', 'x0 x7', 'x0 x8', 'x0 x9', 'x0 x10', 'x0 x11', 'x0 x12', 'x1^2', 'x1 x2', 'x1 x3', 'x1 x4', 'x1 x5', 'x1 x6', 'x1 x7', 'x1 x8', 'x1 x9', 'x1 x10', 'x1 x11', 'x1 x12', 'x2^2', 'x2 x3', 'x2 x4', 'x2 x5', 'x2 x6', 'x2 x7', 'x2 x8', 'x2 x9', 'x2 x10', 'x2 x11', 'x2 x12', 'x3^2', 'x3 x4', 'x3 x5', 'x3 x6', 'x3 x7', 'x3 x8', 'x3 x9', 'x3 x10', 'x3 x11', 'x3 x12', 'x4^2', 'x4 x5', 'x4 x6', 'x4 x7', 'x4 x8', 'x4 x9', 'x4 x10', 'x4 x11', 'x4 x12', 'x5^2', 'x5 x6', 'x5 x7', 'x5 x8', 'x5 x9', 'x5 x10', 'x5 x11', 'x5 x12', 'x6^2', 'x6 x7', 'x6 x8', 'x6 x9', 'x6 x10', 'x6 x11', 'x6 x12', 'x7^2', 'x7 x8', 'x7 x9', 'x7 x10', 'x7 x11', 'x7 x12', 'x8^2', 'x8 x9', 'x8 x10', 'x8 x11', 'x8 x12', 'x9^2', 'x9 x10', 'x9 x11', 'x9 x12', 'x10^2', 'x10 x11', 'x10 x12', 'x11^2', 'x11 x12', 'x12^2']

boston + Ridge vs RandomForestClassifier

```
In [31]: from sklearn.linear_model import Ridge  
ridge = Ridge().fit(X_train_scaled, y_train)  
print("상호작용 특성이 없을 때 점수: {:.3f}".format(ridge.score(X_test_scaled, y_test)))  
ridge = Ridge().fit(X_train_poly, y_train)  
print("상호작용 특성이 있을 때 점수: {:.3f}".format(ridge.score(X_test_poly, y_test)))
```

상호작용 특성이 없을 때 점수: 0.621
상호작용 특성이 있을 때 점수: 0.753

```
In [32]: from sklearn.ensemble import RandomForestRegressor  
rf = RandomForestRegressor(n_estimators=100, random_state=0).fit(X_train_scaled, y_train)  
print("상호작용 특성이 없을 때 점수: {:.3f}".format(rf.score(X_test_scaled, y_test)))  
rf = RandomForestRegressor(n_estimators=100, random_state=0).fit(X_train_poly, y_train)  
print("상호작용 특성이 있을 때 점수: {:.3f}".format(rf.score(X_test_poly, y_test)))
```

상호작용 특성이 없을 때 점수: 0.795
상호작용 특성이 있을 때 점수: 0.773

특성에 상호작용을 추가하면 랜덤 포레스트의 성능은 오히려 감소합니다.
즉 훈련 데이터를 전처리하지 않아도 됩니다.

일변량 비선형 변환

일변량 변환

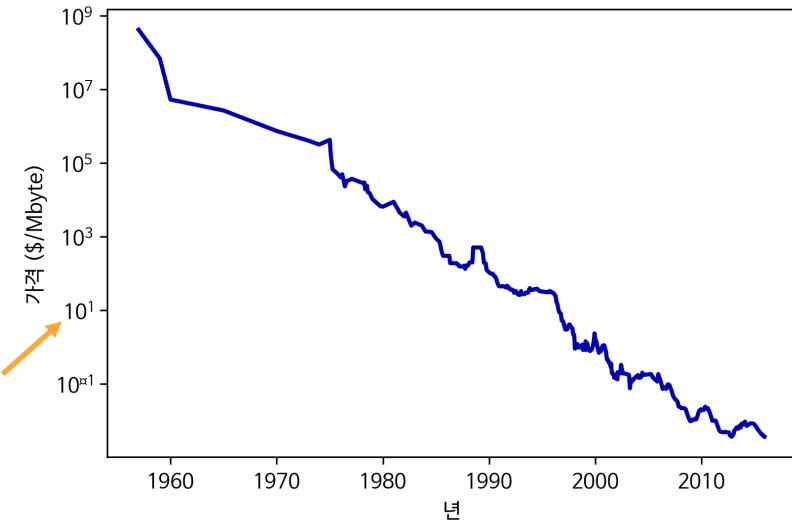
log, exp, sin 같은 수학 함수를 적용하여 특성 값을 변환할 수 있습니다.

선형 모델이나 신경망 모델 같은 경우 데이터 스케일에 민감합니다.

(신경망의 경우 특성 추출에 대해서는 강력하지만 입력 특성의 스케일은 평균 0, 표준 편차 1로 정규화하는 것이 좋습니다)

ex) 2장의 컴퓨터 메모리 가격 데이터 예제

로그 스케일로 변환하여 선형 모델을 적용할 수 있습니다.



카운트 데이터

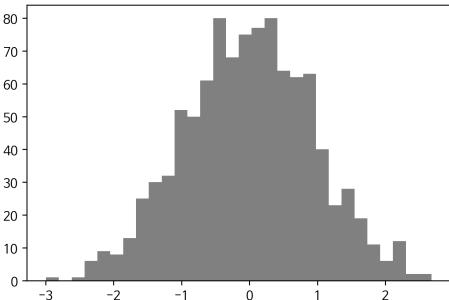
첫 번째 특성의 히스토그램

```
In [33]: rnd = np.random.RandomState(0)
X_org = rnd.normal(size=(1000, 3))
w = rnd.normal(size=3)

X = rnd.poisson(10 * np.exp(x_org))
y = np.dot(X_org, w)
print(X[:10, 0])
```

[56 81 25 20 27 18 12 21 109 7]

랜덤한 가중치를 만들어 정규 분포 데이터에 상응하는 타깃을 생성합니다.
입력 데이터는 푸아송 분포로 바꿉니다.



```
In [34]: print("특성 출현 횟수:\n{}".format(np.bincount(X[:, 0].astype('int'))))
```

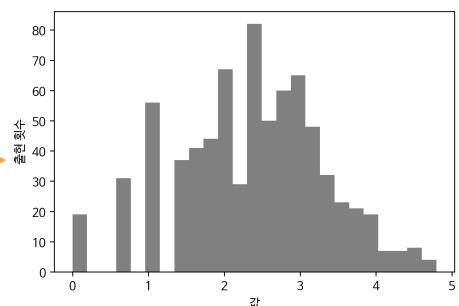
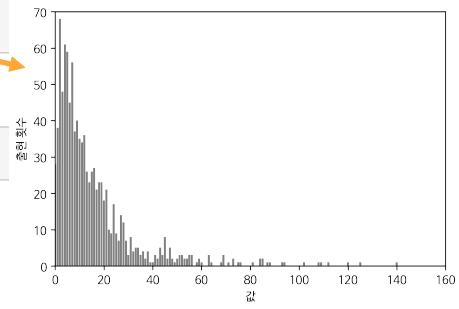
특성 출현 횟수:

28	38	68	48	61	59	45	56	37	40	35	34	36	26	23	26	27	21	23	23	18	21	10	9	17
9	7	14	12	7	3	8	4	5	5	3	4	2	4	1	1	3	2	5	3	8	2	5	2	1
2	3	3	2	2	3	3	0	1	2	1	0	0	3	1	0	0	0	1	3	0	1	0	2	0
1	1	0	0	0	0	1	0	0	2	2	0	1	1	0	0	0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

```
In [37]: X_train_log = np.log(X_train + 1)
X_test_log = np.log(X_test + 1)
```

log(0)을 방지하기 위해 1을 더함

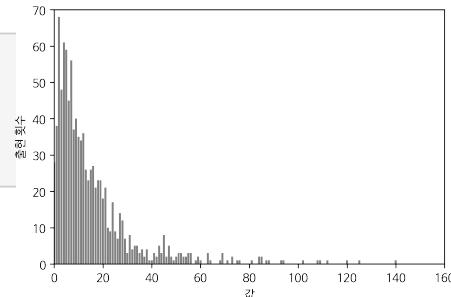
푸아송 분포를 따르는 데이터를
로그 스케일로 바꾸면 정규 분포와
비슷해 집니다.



카운트 데이터 + Ridge

```
In [36]: from sklearn.linear_model import Ridge  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)  
score = Ridge().fit(X_train, y_train).score(X_test, y_test)  
print("테스트 점수: {:.3f}".format(score))
```

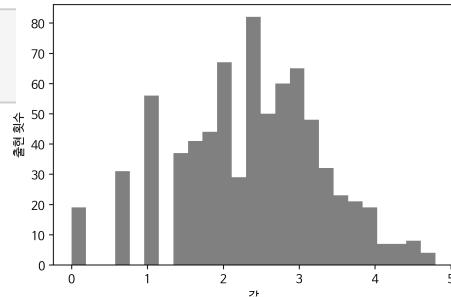
테스트 점수: 0.622



푸아송 분포로 바꾼 데이터에서 훈련한 결과

```
In [39]: score = Ridge().fit(X_train_log, y_train).score(X_test_log, y_test)  
print("테스트 점수: {:.3f}".format(score))
```

테스트 점수: 0.875



로그 스케일로 바꾼 데이터에서 훈련한 결과

특성 변환은 어렵습니다.

데이터와 모델에 맞는 최적의 변환을 찾기가 어렵습니다.

특성마다 변환 방법이 모두 다를 수 있습니다.

트리기반 모델에서는 불필요하지만 선형 모델에서는 필수적입니다(특히 특성의 개수가 적을 때).

SVM이나 신경망 같은 모델은 특성 추가 보다는 특성의 스케일을 맞추는 것이 중요합니다.

SVM은 커널 기법으로 무한 다항 차원을 매핑하는 효과를 낼 수 있으며, 신경망은 데이터에서 유용한 특성을 만들 수 있습니다(표현 학습representation learning).

특성 자동 선택

특성 선택

새로운 특성을 만들어 추가하면 모델이 복잡해지고 과대적합될 가능성이 높습니다.

특성이 추가되거나 고차원인 데이터셋에서는 불필요한 특성을 제외하면 모델이 간단해지고 일반화 성능을 높일 수 있습니다.

→ 일변량 통계 기반 선택, 모델 기반 선택, 반복적 선택 방법이 있습니다.

- 타깃 값을 사용합니다.
- 훈련 세트와 테스트 세트로 나눈 다음 훈련 세트만 사용해야 합니다(6장 정보 누설).

일변량 통계

개개의 특성과 타깃 사이에 관계가 높은 특성을 선택합니다.

분류: 분산 분석(ANOVA^{analysis of variance})은 클래스별 평균을 비교합니다. F 값이 크면 그 특성의 클래스별 평균이 서로 다르다는 의미입니다(타깃 예측에 유용함).

$$F = \frac{SS_{between}/(k - 1)}{(SS_{tot} - SS_{between})/(n - k)}$$

클래스별 평균 분산

$$SS_{between} = \sum_{j=1}^k n_j (\bar{x}_j - \bar{x})^2, SS_{tot} = \sum_{i=1}^n (x_i - \bar{x})^2$$

전체 분산

SelectKBest나 SelectPercentile에서 **f_classif**로 지정

일변량 통계

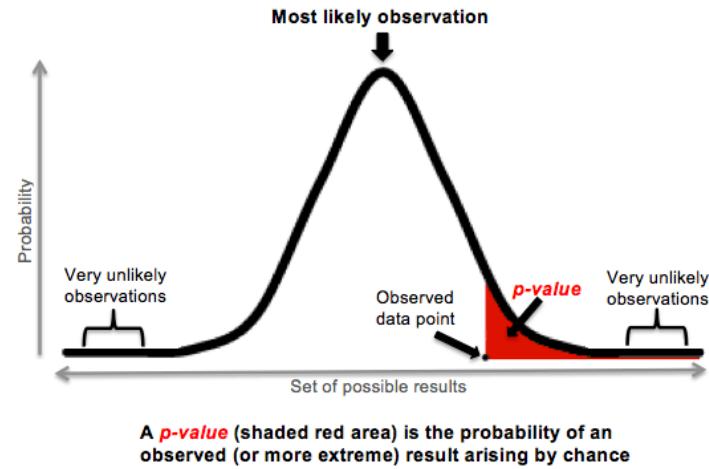
회귀: 상관계수를 이용하여 F값을 계산합니다.

$$Corr = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{std(x)std(y)}$$

SelectKBest나 SelectPercentile에서 **f_regression**로 지정

F-분포 distribution에서 계산된 F값(scores_ 속성)의 이후의 면적을
p-값(pvalue(pvalues_ 속성)이라 합니다.

p-값이 크면(F값이 작으면) 타깃에 미치는 영향이 작다고
판단합니다(기본 p-값은 0.05)



cancer + noise

```
In [40]: from sklearn.datasets import load_breast_cancer
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import train_test_split

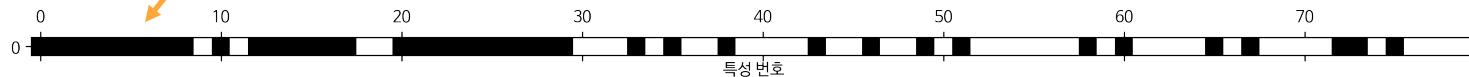
cancer = load_breast_cancer()

# 고정된 난수를 발생시킵니다
rng = np.random.RandomState(42)
noise = rng.normal(size=(len(cancer.data), 50))
# 데이터에 노이즈 특성을 추가합니다
# 처음 30개는 원본 특성이고 다음 50개는 노이즈입니다
X_w_noise = np.hstack([cancer.data, noise])

X_train, X_test, y_train, y_test = train_test_split(
    X_w_noise, cancer.target, random_state=0, test_size=.5)
# f_classif(기본값)과 SelectPercentile을 사용하여 특성의 50%를 선택합니다
select = SelectPercentile(score_func=f_classif, percentile=50)
select.fit(X_train, y_train)
# 훈련 세트에 적용합니다
X_train_selected = select.transform(X_train)

print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

훈련세트에만 적용



절반을 선택
SelectKBest():
k개의 특성을 선택

X_train.shape: (284, 80)
X_train_selected.shape: (284, 40)

SelectPercentile + LogisticRegression

성능 향상을 꾀할 수 있고 모델을 해석하기 쉬어 집니다.

```
In [42]: from sklearn.linear_model import LogisticRegression

# 테스트 데이터 변환
X_test_selected = select.transform(X_test)

lr = LogisticRegression()
lr.fit(X_train, y_train)
print("전체 특성을 사용한 점수: {:.3f}".format(lr.score(X_test, y_test)))
lr.fit(X_train_selected, y_train)
print("선택된 일부 특성을 사용한 점수: {:.3f}".format(
    lr.score(X_test_selected, y_test)))
```

전체 특성을 사용한 점수: 0.930

선택된 일부 특성을 사용한 점수: 0.940

노이즈 때문에 성능이 저하됨
(특성이 너무 많거나 일부 특성이 도움이 안된다고 확신이 들 경우 적용)

모델 기반 선택

feature_importances_(결정트리)나 coef_(선형모델) 값을 사용합니다.

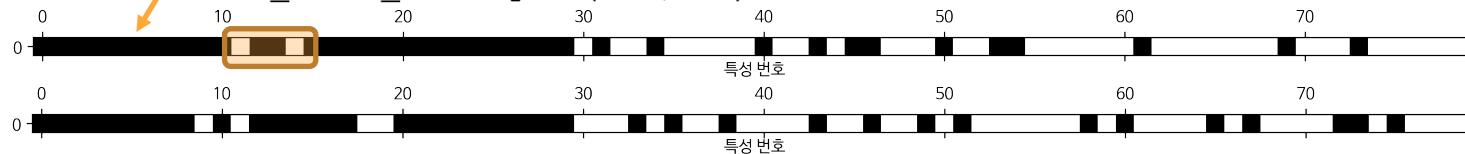
기본 임계값 : L1 페널티(라쏘)가 있는 경우 10^{-5} , 그외는 평균값

mean
1.3*median

```
In [43]: from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(
    RandomForestClassifier(n_estimators=100, random_state=42),
    threshold="median")
```

```
In [44]: select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
```

X_train.shape: (284, 80)
X_train_l1.shape: (284, 40)



SelectFromModel + LogisticRegression

특성 선택을 위한 모델(RandomForestClassifier)과 학습 모델(LogisticRegression)이 다를 수 있습니다.

```
In [46]: X_test_11 = select.transform(X_test)
score = LogisticRegression().fit(X_train_11, y_train).score(X_test_11, y_test)
print("Test score: {:.3f}".format(score))
```

Test score: 0.951

```
lr.fit(X_train_selected, y_train)
print("선택된 일부 특성을 사용한 점수: {:.3f}".format(
    lr.score(X_test_selected, y_test)))
```

전체 특성을 사용한 점수: 0.930
선택된 일부 특성을 사용한 점수: 0.940

4개 특성을 못 잡아 낸 SelectPercentile
보다 성능이 상승함

반복적 특성 선택

특성을 하나씩 추가하면서 모델을 만들거나, 모든 특성에서 하나씩 제거하면서 모델을 만듭니다.

여러개의 모델을 만들기 때문에 계산 비용이 많이 듭니다.

scikit-learn은 재귀적 특성 제거(RFE^{recursive feature elimination})를 제공하며, 전체 특성을 포함한 모델에서 지정된 개수만큼 남을 때까지 특성 중요도가 가장 낮은 특성을 제거해 나갑니다.

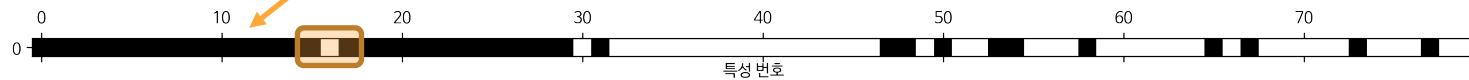
`feature_importances_`(결정트리)나 `coef_`(선형모델) 값을 사용합니다.

회귀 모델의 전진 선택법(forward stepwise selection)과 후진 선택법(backward stepwise selection)은 제공하지 않습니다. 특성을 추가/제거하면서 직접 R2 점수를 가지고 계산할 수 있습니다.

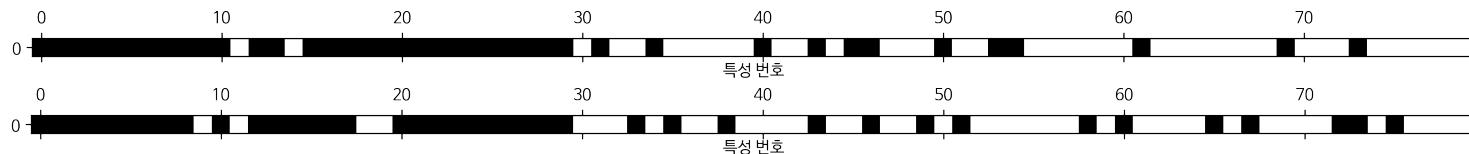
RFE

```
In [47]: from sklearn.feature_selection import RFE  
select = RFE(RandomForestClassifier(n_estimators=100, random_state=42),  
               n_features_to_select=40)  
  
select.fit(X_train, y_train) # 선택된 특성을 표시합니다  
mask = select.get_support()
```

SelectFromModel에서와 동일한 RandomForestClassifier를 사용



40개의 RandomForestClassifier 모델을
만듭니다(시간이 오래 걸림)



RFE + LogisticRegression

RFE에 사용한 모델을 이용해 평가에 사용할 수 있습니다.

```
In [48]: X_train_rfe = select.transform(X_train)
X_test_rfe = select.transform(X_test)

score = LogisticRegression().fit(X_train_rfe, y_train).score(X_test_rfe, y_test)
print("테스트 점수: {:.3f}".format(score))

테스트 점수: 0.951
```

RFE(RandomForestClassifier())

```
In [49]: print("테스트 점수: {:.3f}".format(select.score(X_test, y_test)))
```

테스트 점수: 0.951

로지스틱 회귀와 랜덤 포레스트의 성능이 비슷함

- 특성 선택이 좋으면 로지스틱 회귀의 성능이 랜덤 포레스트와 견줄 수 있음
- 랜덤 포레스트를 사용하면 특성 선택의 필요성이 많이 감소됨

전문가 지식 활용

특성 공학과 도메인 지식

특성 공학은 애플리케이션마다 다른 전문가의 지식을 사용할 수 있는 영역입니다.

머신러닝은 규칙 기반 시스템과 다르지만 분야의 전문 지식은 여전히 유효합니다.

예) 항공료를 잘 예측하려면 날짜, 항공사, 출발지, 도착지외에 그 지역의
공휴일(양/음력)이나 휴가 기간을 잘 알아야 합니다.

애플리케이션마다 내재된 지식이 특성으로 추가되어도 알고리즘이 반드시
사용하는 것은 아닙니다.

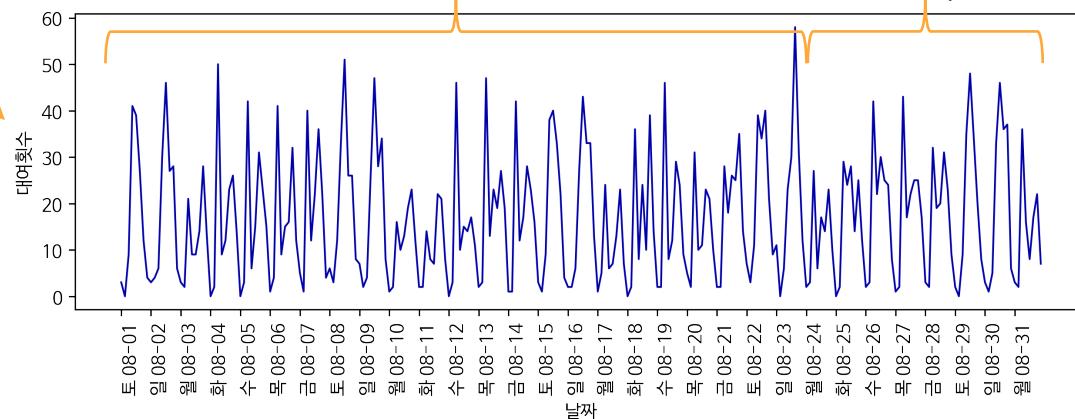
하지만 사용되지 않더라도 특성을 추가하는 것이 문제가 되지는 않습니다.

특성 공학과 도메인 지식

뉴욕의 시티 바이크 대여 예측

(<https://www.citibikenyc.com/system-data>)

3시간 간격의
대여 횟수
(타깃)



```
In [50]: citibike = mglearn.datasets.load_citibike()
```

```
In [51]: print("Citibike data:\n{}".format(citibike.head()))
```

```
Citibike data:  
starttime  
2015-08-01 00:00:00      3.0  
2015-08-01 03:00:00      0.0  
2015-08-01 06:00:00      9.0  
2015-08-01 09:00:00     41.0  
2015-08-01 12:00:00     39.0  
Freq: 3H, Name: one, dtype: float64
```

Unix Time + RandomForestRegressor

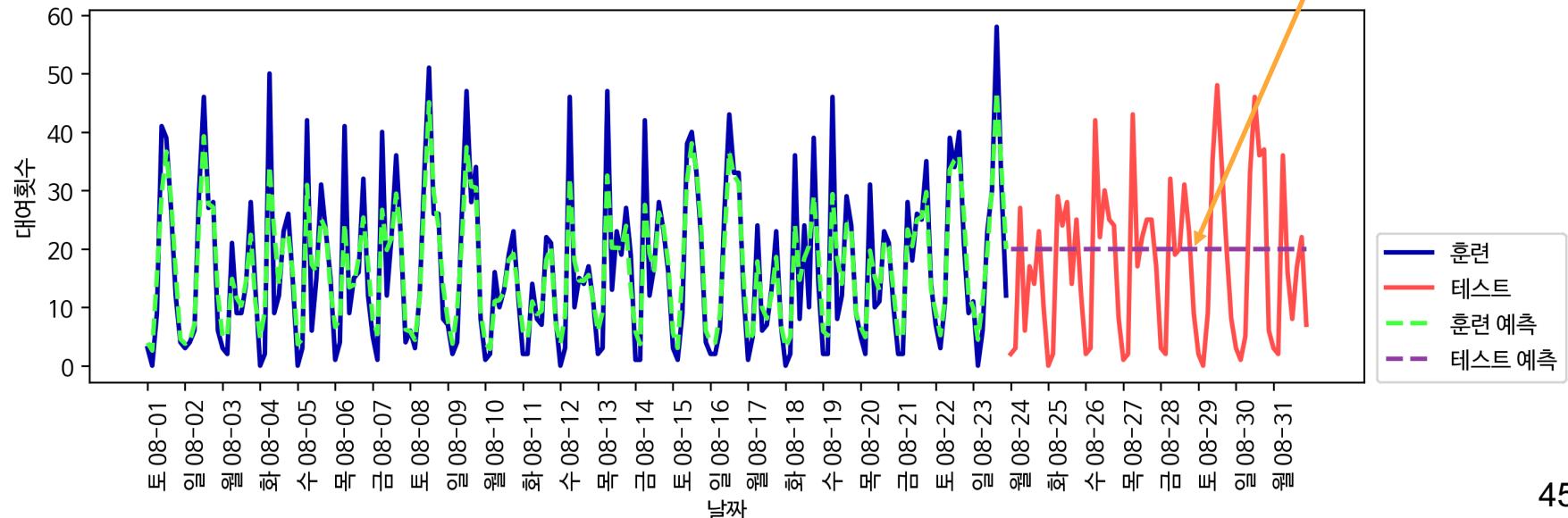
```
In [55]: regressor = RandomForestRegressor(n_estimators=100, random_state=0)  
eval_on_features(X, y, regressor)
```

훈련 데이터와 테스트
데이터로 분리하여
모델 평가

테스트 세트 $R^2: -0.04$

[248, 1]

트리모델의 특징
(외삽 불능)



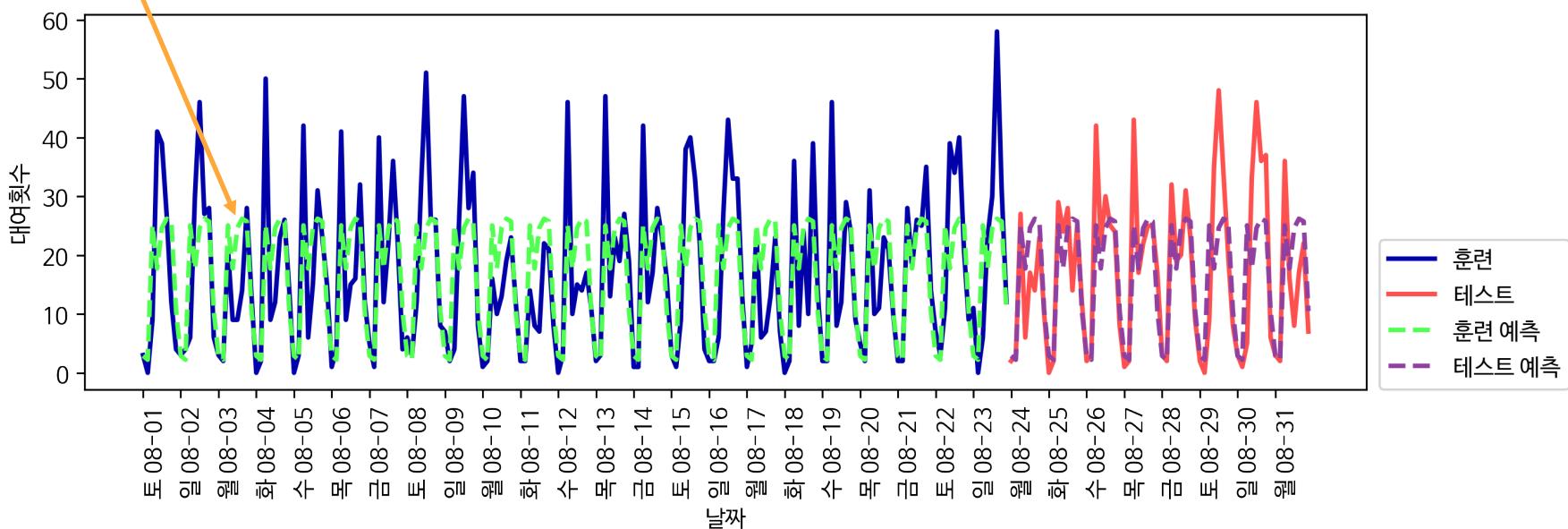
Hour + RandomForestRegressor

시간 데이터만
있으므로 매일 동일한
패턴이 학습됨

```
In [56]: X_hour = citibike.index.hour.values.reshape(-1, 1)
eval_on_features(X_hour, y, regressor)
```

테스트 세트 R^2 : 0.60

[248, 1]



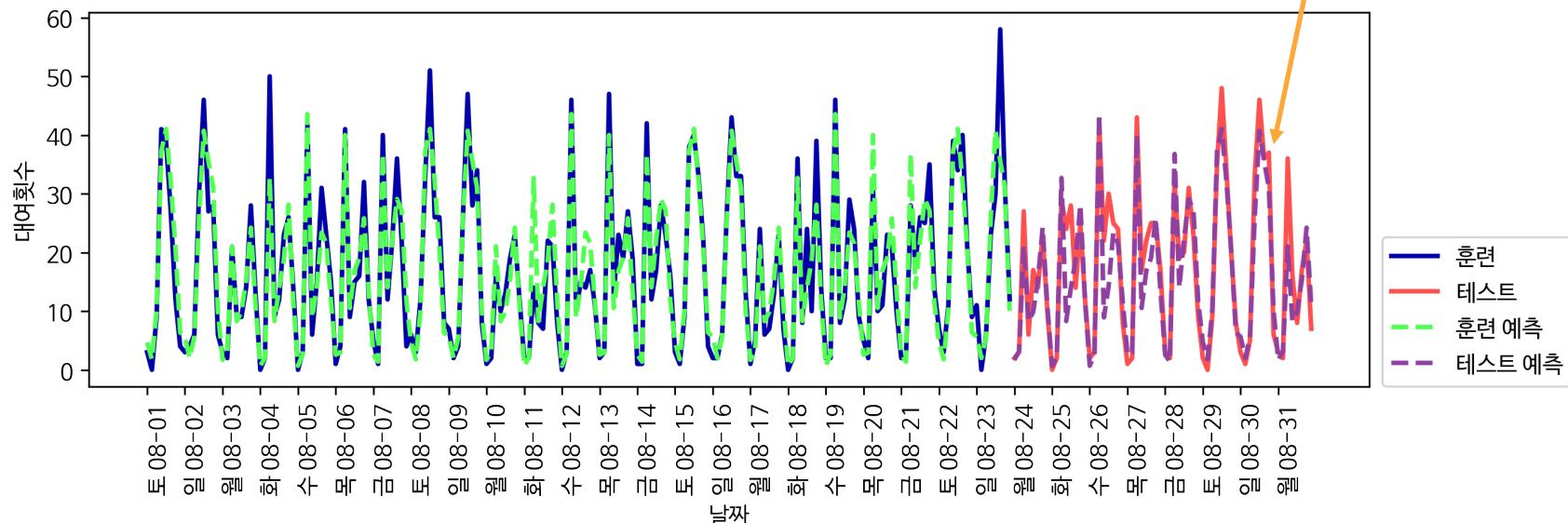
Hour,week + RandomForestRegressor

```
In [57]: X_hour_week = np.hstack([citibike.index.dayofweek.values.reshape(-1, 1),  
                                citibike.index.hour.values.reshape(-1, 1)])  
eval_on_features(X_hour_week, y, regressor)
```

테스트 세트 $R^2: 0.84$

[248, 2]

요일/시간 데이터를 사용하여
주간 패턴을 학습함

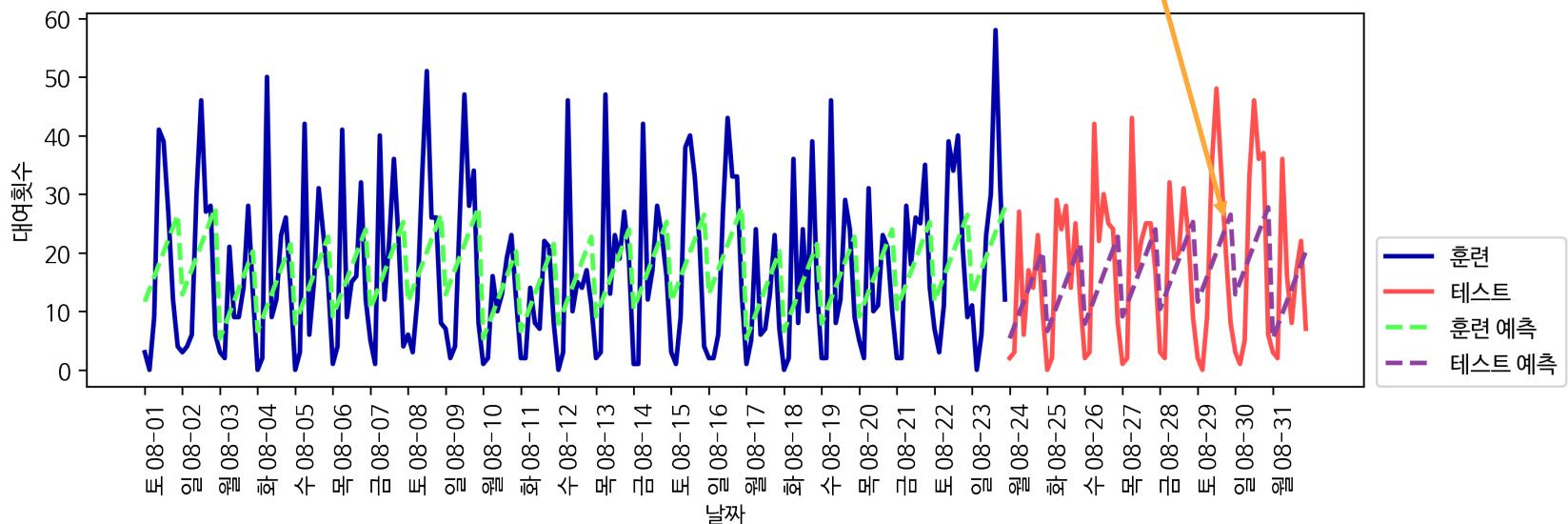


Hour,week + LinearRegression

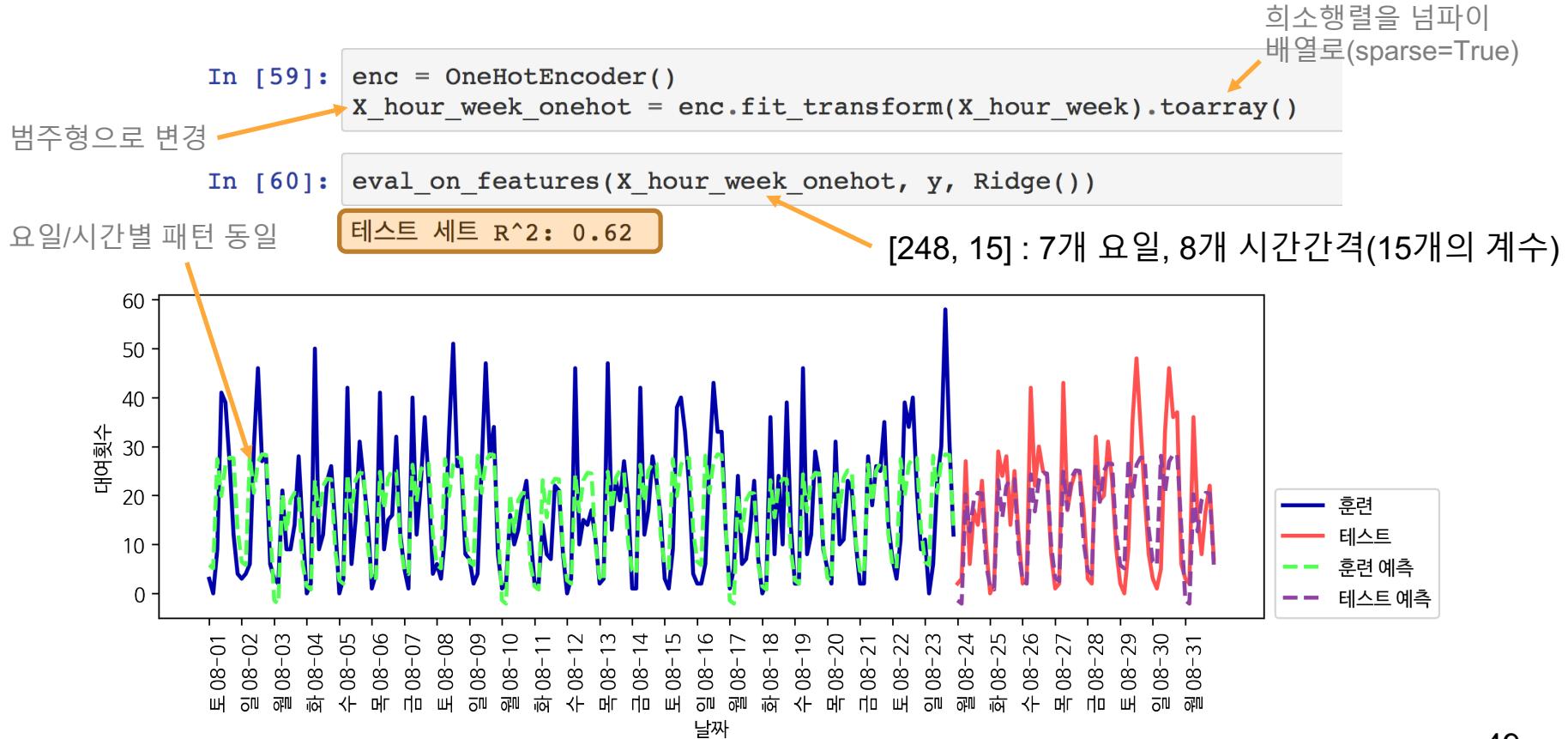
```
In [58]: from sklearn.linear_model import LinearRegression  
eval_on_features(X_hour_week, y, LinearRegression())
```

테스트 세트 $R^2: 0.13$

요일,시간이 정수라 연속형으로
인식합니다.(2개의 계수)



OneHotEncoder + LinearRegression



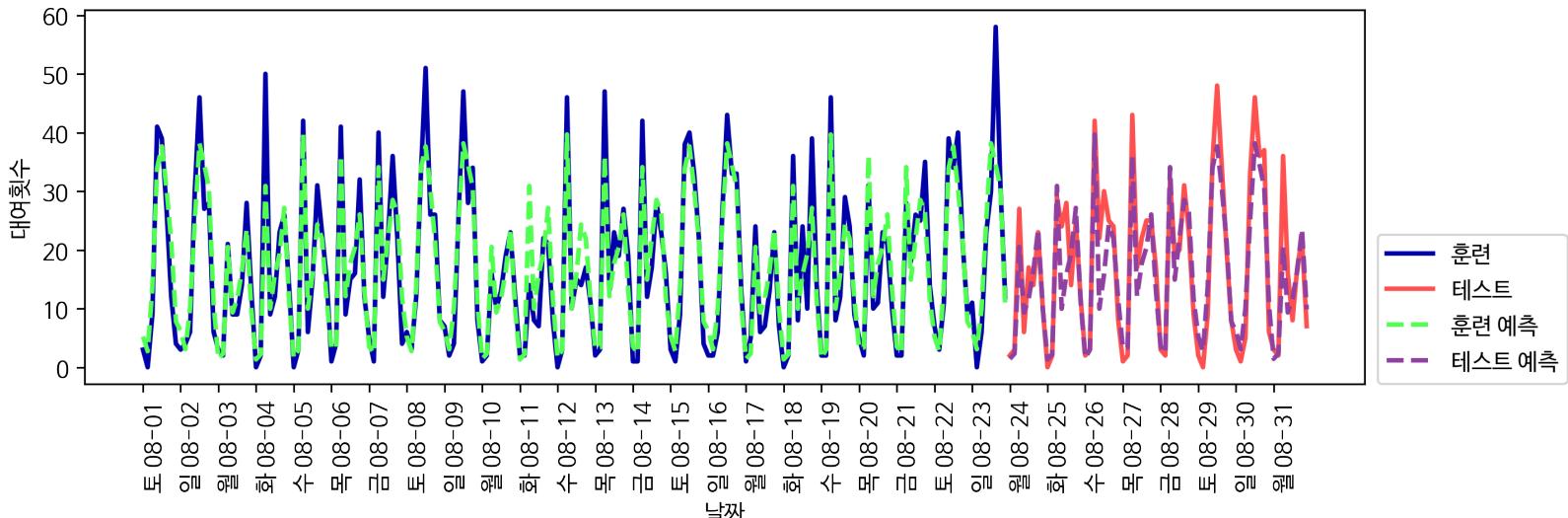
PolynomialFeatures + LinearRegression

```
In [61]: poly_transformer = PolynomialFeatures(degree=2, interaction_only=True,  
                                         include_bias=False)  
X_hour_week_onehot_poly = poly_transformer.fit_transform(X_hour_week_onehot)  
lr = Ridge()  
eval_on_features(X_hour_week_onehot_poly, y, lr)
```

랜덤 포레스트와
비슷한 수준의
성능

테스트 세트 $R^2: 0.85$

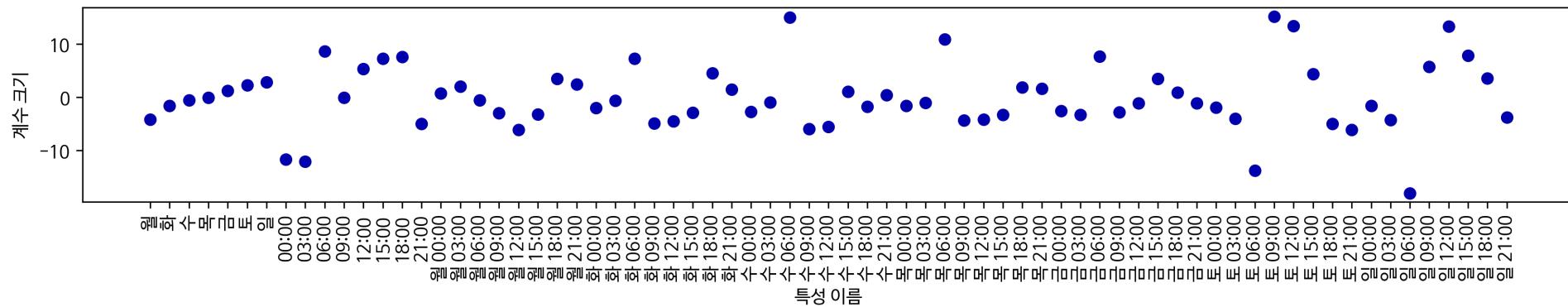
중복을 제외한 조합
 $[248, 120] : \binom{15}{2} + 15 = 105 + 15 = 120$



LinearRegression's coef_

선형 모델은 랜덤 포레스트와는 달리 학습된 모델 파라미터를 확인할 수 있습니다.

```
coef_nonzero = lr.coef_[lr.coef_ != 0]
plt.plot(coef_nonzero, 'o')
```



감사합니다.

-질문-