비타민 5주차 복습세션

데이터 표현과 특성공학

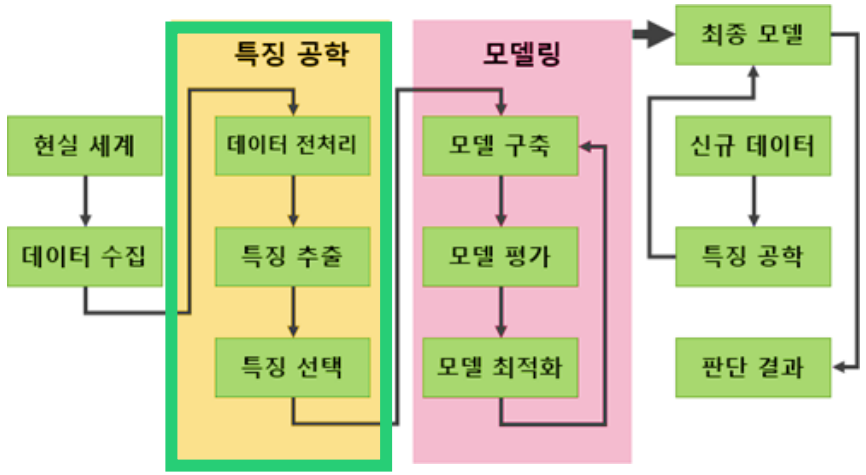
3조 윤희재 이승재 이원재 장윤서

목차

01. 특성 공학의 개념

01. 특성공학의 개념

특성공학(Feature engineering)



머신러닝 단계도

◆ 특성공학이란?

모델 훈련에 사용하기에 가장 적합한 데이터 특성(표현)을 찾는 단계

◆ 특성이란?

Feature, 모델 input 데이터에서 변수로 나타내어짐

원본 데이터의 변수는 물론 이를 가공한 새 변수도 특성이 될 수 있음

◆ 특성공학의 배경

모델에서 가정한 데이터의 특성과 실제 수집된 데이터의 특성 간의 괴리가 있음

ex. 모델에는 연속형 변수를 입력해줘야 하는데 실제 수집된 데이터는 범주형 변수인 경우 또는 원본 데이터의 측정 단위(스케일)이 일치하지 않는 경우

모델에 적합한 데이터 표현이 지도 학습 모델에서 매개변수를 선택하는 것보다 성능에 더 큰 영향을 미침

특성공학의 두 단계

특성생성(특성추출)

원본 데이터의 특성을 모델에 적합한 표현방식으로 바꾸거나

여러 특성을 합쳐서 더 유용한 특성을 만듬

특성선택

추출된 특성 중 가장 유용한 조합을 선택

02. 특성생성 기법

02. 특성생성 기법

특성을 생성하는 방법들

- 범주형 변수의 변환

  범주형 특성 ▶ 모델에 맞는 새 특성 (더미변수 등) 생성

- 선형모델에서의 구간분할(이산화)

  연속형 특성 ▶ 특정 범위로 그룹화해 이산형 특성 생성

- 상호작용과 다항식

  원본 특성 X ▶ X의 다항식을 새 특성으로 사용 (다항회귀)

- 일변량 비선형 변환

  원본 특성 X ▶ X에 log, exp, sin 함수를 적용해 새 특성 생성

02. 특성생성 기법

범주형 변수의 변환

◆ 짚고 넘어가기

연속형 변수

수치형 변수, 연속적인 값을 가짐

ex. 키, 몸무게, 온도, 나이, 고객수, 구매율

범주형 변수

문자 또는 숫자로 이루어져 있는 변수, 범주(카테고리)가 있으며 요인변수라고도 부른다

ex. 학년(1,2,3), 성별(남,여)

◆ 범주형 변수의 종류

순서가 없는 요인변수

- **문자형 요인변수**

  ex. 직장(공기업, 사기업, 자영업)

- **숫자형 요인변수**

  용량을 줄이거나 데이터 취합 방식에 따라 문자형 요인변수가 숫자로 인코딩 된 경우

  ex. 직장(1: 공기업, 2,: 사기업, 3: 자영업)

순서가 있는 요인변수

요인의 수준이 순서를 갖는 변수, 순서 범주형 변수라고도 불림, 범주형 변수로 다루면 순서에 대한 정보를 누락할 수 있기 때문에 수치형 변수로 변환해 사용

ex. 정도(1: 매우나쁨 2: 나쁨 3: 보통 4: 좋음 5: 매우좋음)

문자형 요인변수 변환 - 더미변수

? 범주형 변수로 회귀모델을 돌릴 수 있을까?

No. 앞서 배운 회귀모델은 연속형(수치형) 변수를 학습데이터로 가정한 모델이기 때문에 범주형 변수를 넣을 수 없다

더미변수

데이터를 사용하기 위해 범주형 변수를 연속형 변수스럽게 표현한 변수, 가변수라고도 부름

문자형 요인변수를 더미변수로 변환하기

데이터셋 : 미국 성인 소득 데이터 (adult)

선택된 변수:

age, workclass, gender

```
data = data[['age', 'workclass','gender']]
```

```
data.head()
```

	age	workclass	gender
0	39	State-gov	Male
1	50	Self-emp-not-inc	Male
2	38	Private	Male
3	53	Private	Male
4	28	Private	Female

02. 특성생성 기법 – 범주형 변수의 변환

문자형 요인변수를 더미변수로 변환하기

Step1. 전처리단계 - 문자열로 된 범주형 데이터 확인하기

열에 어떤 의미있는 범주형 데이터가 있는지 확인. 사용자로부터 입력받은 데이터를 다룰 때는 특히 이상치 값;정해진 범주 밖의 값이 있거나 대소문자가 틀린 경우(male – man, Male) 이 있을 수 있어 확인이 필요함 (value_counts)

```
print(data.workclass.value_counts())
```

Step2. 원-핫-인코딩

get_dummies(data) : 데이터 내 문자형을 가진 범주형 변수를 자동으로 더미변수로 변환

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

02. 특성생성 기법 – 범주형 변수의 변환

더미변수의 특성

◆ 원본특성 'workclass'의 원-핫-인코딩 후 생성된 특성:

Workclass는 9개의 value값을 가졌으므로 0, 1로 이루어진 9개의 특성(열)이 생성됨

```
['workclass_ ?',
 'workclass_ Federal-gov',
 'workclass_ Local-gov',
 'workclass_ Never-worked',
 'workclass_ Private',
 'workclass_ Self-emp-inc',
 'workclass_ Self-emp-not-inc',
 'workclass_ State-gov',
 'workclass_ Without-pay']
```

	age	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	...	workclass_ State-gov	workclass_ Without-pay
0	39	0	0	0	...	1	0
1	50	0	0	0	...	0	0
2	38	0	0	0	...	0	0
3	53	0	0	0	...	0	0
4	28	0	0	0	...	0	0

◆ 더미변수의 다중공선성 문제

Workclass로부터 생성된 9개 이진변수의 합은 항상 1 :

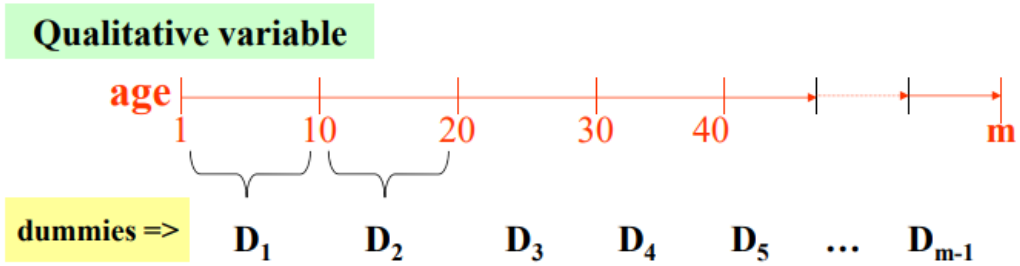
8개의 이진변수의 값을 정의하고 나면 9번째 변수의 값을 알 수 있음

> 다중공선성 문제 발생

02. 특성생성 기법 – 범주형 변수의 변환

더미변수의 다중공선성 문제 해결

다중공선성을 피하기 위해 요인변수가 m개 개별 수준을 가질 경우
m-1개의 더미변수만 모델에 도입



◆ drop_first 옵션

drop_first=True 옵션을 넣어 맨 첫번째 더미변수를 제거하면 다중공선성 문제를 해결할 수 있음

```
data_dummies = pd.get_dummies(data, drop_first=True)
```

숫자형 요인변수 변환

예시 데이터

```
demo_df = pd.DataFrame({'숫자 특성': [0, 1, 2, 1],
                        '범주형 특성': ['양말', '여우', '
양말', '상자']})
```

예시 데이터:
demo_df:

	숫자 특성	범주형 특성
0	0	양말
1	1	여우
2	2	양말
3	1	상자

get_dummies는 문자열 특성만 인코딩하기 때문에 숫자 특성은 변환할 수 없음
숫자형 특성을 더미변수로 변환하려면 해당 변수를 문자열로 바꿔준 후 columns 매개변수에 인코딩하고 싶은 열을 명시해야 함

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

Scikit-learn으로 범주형 변수 다루기

◆ OneHotEncoder 클래스 사용

Step1. import

```
from sklearn.preprocessing import OneHotEncoder
```

Step2. oneHotEncoder 객체생성

```
# sparse=False로 설정하면 OneHotEncode가 희소 행렬이 아니라 넘파이 배열을 반환
ohe = OneHotEncoder(sparse=False)
```

Step3. fit_transform

```
print(ohe.fit_transform(demo_df)
print(ohe.get_feature_names_out())
```

생성된 특성:

['숫자 특성_0' '숫자 특성_1' '숫자 특성_2' '범주형 특성_상자' '범주형 특성_양말' '범주형 특성_여우']

예시 데이터:

demo_df:

	숫자 특성	범주형 특성
0	0	양말
1	1	여우
2	2	양말
3	1	상자

결과 출력:

```
[[1. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 1.]
 [0. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 0.]]
```

Scikit-learn으로 범주형 변수 다루기

```
ColumnTransformer([
        ("객체명", 함수(), ["변수명"]),
        ("객체명", 함수(), ["변수명"])        ])
```

◆ OneHotEncoder + ColumnTransformer

OneHotEncoder는 get_dummies와 다르게 모든 변수를 범주형으로 가정하고 더미변수로 변환
따라서 연속형 변수가 섞인 data에 바로 적용할 수 없음
ColumnTransformer 클래스는 입력 열마다 다른 변환을 적용할 수 있게 해 이러한 문제를 해결

Step1. import

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler #표준화 클래스
```

Step2. ColumnTransformer 객체생성

```
ct = ColumnTransformer(                                    # 연속형 변수: 표준화 적용
    [("scaling", StandardScaler(), ['age', 'hours-per-week']),
     ("onehot", OneHotEncoder(sparse=False),   # 범주형 변수: 원-핫-인코딩 적용
      ['workclass', 'education', 'gender', 'occupation'])])
```

Scikit-learn으로 범주형 변수 다루기

Step3. 데이터 준비하기

```
from sklearn.model_selection import train_test_split
```

학습 데이터 (data): 미국 성인 소득 데이터 / 다른 변수들로 income 예측

	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
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K

```
# input=data_features target="income"
data_features = data.drop("income", axis=1)
# train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data_features, data.income, random_state=0)
```

Scikit-learn으로 범주형 변수 다루기

Step4. fit & transform

X_train 데이터를 앞에서 만든 ct 객체에 fit & transform 하기

```
X_train_trans = ct.fit_transform(X_train)
X_test_trans = ct.fit_transform(X_test)
```

Step5. 모델 학습

타겟인 income 변수가 범주형 변수이므로 로지스틱 회귀(분류) 모델을 사용

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_trans, y_train)
#테스트 점수 출력
logreg.score(X_test_trans, y_test)))
```

테스트 점수:
0.81

income
<=50K
<=50K
<=50K
<=50K
<=50K

회귀식에서 더미변수가 미치는 영향

- 더미변수는 회귀식에서 해당 변수의 효과를 상수값(절편)으로 만들어 회귀 그래프를 평행이동시킴

더미변수를 적용한 회귀식 :

$$y_t = \beta_1 + \beta_2 X_t + \beta_3 G_t + \varepsilon_t$$

For male: $G_t = 1$.
For female: $G_t = 0$.

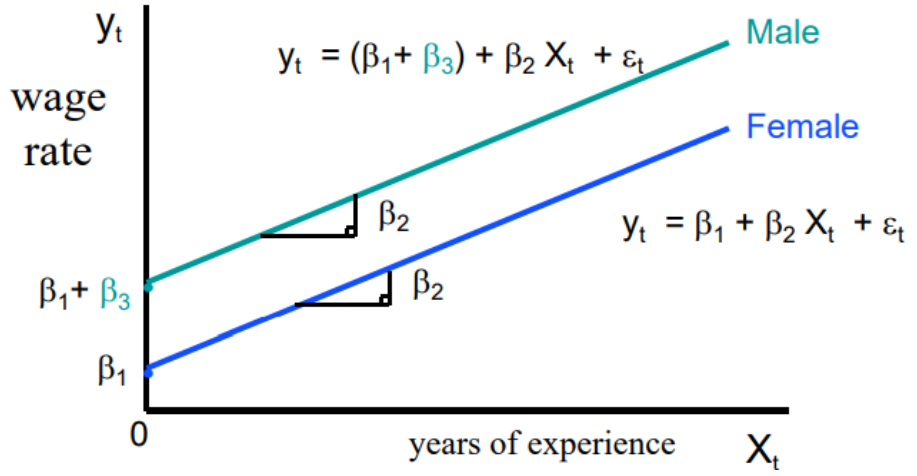
$y_t = (\beta_1 + \beta_3) + \beta_2 X_t + \varepsilon_t$: Male Workers

$y_t = \beta_1 + \beta_2 X_t + \varepsilon_t$: Female Workers

y_t = wage rate per hour

X_t = years of experience

Gt: 범주형 변수 gender이 변환된 더미변수



다수의 수준을 갖는 범주형 변수의 특성변환

어떤 요인변수는 가능한 수준의 수가 많아 더미변수로 바꾼다면 데이터 크기가 너무 커질 수 있음. 이러한 경우 요인변수의 수준을 n개 그룹으로 통합하는 것이 방법이 될 수 있다.

실습 문제

예시 데이터셋 : 킹 카운티 지역의 주택 매매 데이터

선택된 변수: ZipCode

	98038	98103	98042	98115	98117	98052	98034	98033	98059	98074	...	98051	98024	98354	98050	98057	98288	98224	98068	98113	98043
ZipCode	788	671	641	620	619	614	575	517	513	502	...	32	31	9	7	4	4	3	1	1	1

1 rows × 80 columns

킹 카운티의 주택 매매 데이터의 ZipCode 변수에는 80개의 개별 수준이 존재한다. ZipCode는 주택 가격에 대한 위치의 효과를 볼 수 있는 중요한 변수이지만 모든 수준을 포함하려면 자유도 79에 해당하는 79의 회귀 계수가 필요하다. 이에 대안으로 우편번호를 5개 기준으로 통합하려고 한다.

다수의 수준을 갖는 범주형 변수의 특성변환

◆ 잔차의 중간값을 기준으로 범주형 변수 그룹화하기

Step1. 초기 회귀모델 만들기

```
from sklearn.linear_model import LinearRegression

predictors = ['SqFtTotLiving', 'SqFtLot', 'Bathrooms',
              'Bedrooms', 'BldgGrade']  #예측변수
outcome = 'AdjSalePrice' #타겟

house_lm = LinearRegression()
house_lm.fit(house[predictors], house[outcome])
```

다수의 수준을 갖는 범주형 변수의 특성변환

생성한 데이터프레임:

	ZipCode	residual
1	98002	-123750.814194
2	98166	-59145.413089
3	98166	190108.725716
4	98168	-198788.774412
5	98168	-91774.996129

◆ 잔차의 중간값을 기준으로 범주형 변수 그룹화하기

Step2. ZipCode 행별로 잔차 저장하기

ZipCode 변수와 앞서 학습한 회귀모델 house_lm의 잔차를 한 데이터프레임으로 묶어준다

```
Zip_resd=pd.DataFrame({
        'ZipCode': house['ZipCode'],
        'residual' : house[outcome]-house_lm.predict(house[predictors])})
```

Step3. ZipCode 수준별로 잔차의 평균 계산하기

데이터 프레임을 ZipCode 변수로 그룹화해 lamda 함수를 적용해준다. Lamda 함수 내에서 ZipCode의 수준별 중간값을 구해 새 변수 median_resicual 로 저장한다.

```
zip_groups = pd.DataFrame([
    *Zip_resd.groupby(['ZipCode']).apply(lambda x: {
        'ZipCode': x.iloc[0,0], 'count': len(x),
        'median_residual': x.residual.median()})
```

다수의 수준을 갖는 범주형 변수의 특성변환

◆ 잔차의 중간값을 기준으로 범주형 변수 그룹화하기

Step4. median_residual 이 높은 순으로 5개 그룹으로 분할

앞서 만든 zip_groups 데이터프레임을 median_residual 기준으로 정렬 후 정렬한 데이터를 5개 그룹으로 통합한 ZipGroup 특성 생성.

```
zip_groups.sort_values('median_residual')
zip_groups['cum_count'] = np.cumsum(zip_groups['count'])
zip_groups['ZipGroup'] = pd.qcut(zip_groups['cum_count'], 5, labels=False, r
etbins=False)
```

Step5. 원본 데이터에 생성된 특성 합치기, 카테고리화

```
to_join = zip_groups[['ZipCode', 'ZipGroup']].set_index('ZipCode')
house = house.join(to_join, on='ZipCode')
house['ZipGroup'] = house['ZipGroup'].astype('category')
```

다수의 수준을 갖는 범주형 변수의 특성변환

◆ 잔차의 중간값을 기준으로 범주형 변수 그룹화하기

결과출력

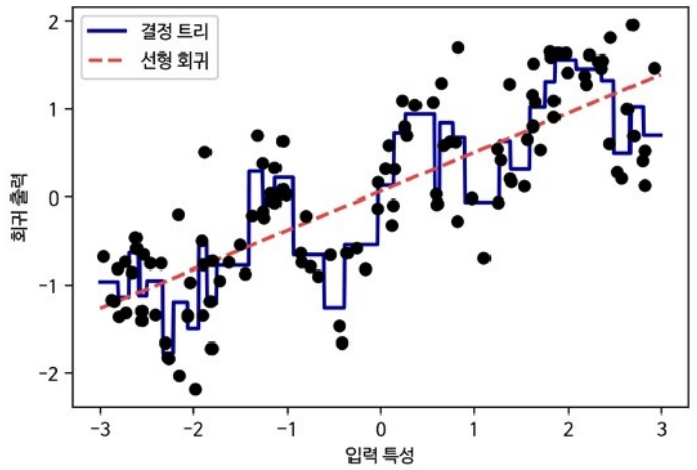
80개의 ZipCode 수준을 16개씩 5개 그룹에 저장한 범주형 특성이 만들어짐

```
print(zip_groups.ZipGroup.value_counts().sort_index())
```

```
0  16
1  16
2  16
3  16
4  16
Name: ZipGroup, dtype: int64
```

선형모델에서의 구간분할(이산화)

일반적으로 선형모델은 선형관계로만 모델링하므로 비선형 데이터에 적합이 어렵다는 한계가 있다.



결정트리 모델과 비교한 선형 회귀 모델의 데이터 적합

결정트리를 사용하기 어려운 고용량/고차원 데이터의 경우 연속형 특성을 구간별로 나눠 그룹화한 특성을 선형모델에 도입하면 예측율을 높일 수 있음

◆ KBinsDiscretizer 클래스

연속형 데이터를 균일한 너비로 분할해주는 함수가 구현됨

```
from sklearn.preprocessing import KBinsDiscretizer
                                # n_bins : 데이터를 나눌 그룹 개수
kb = KBinsDiscretizer(n_bins=10, strategy='uniform')
X_binned = kb.fit_transform(X)    # strategy : 데이터를 나누는 방법
print("bin edges: \n", kb.bin_edges_)
```

연속형 데이터가 나눠진 구간값 출력:

```
bin edges: [array([-2.967, -2.378, -1.789, -1.2 , -0.612,
-0.023, 0.566, 1.155, 1.744, 2.333, 2.921])
```

선형모델에서의 구간분할(이산화)

◆ 인코딩 결과

KBinsDiscretizer은 기본적으로 구간에 원-핫-인코딩을 사용함

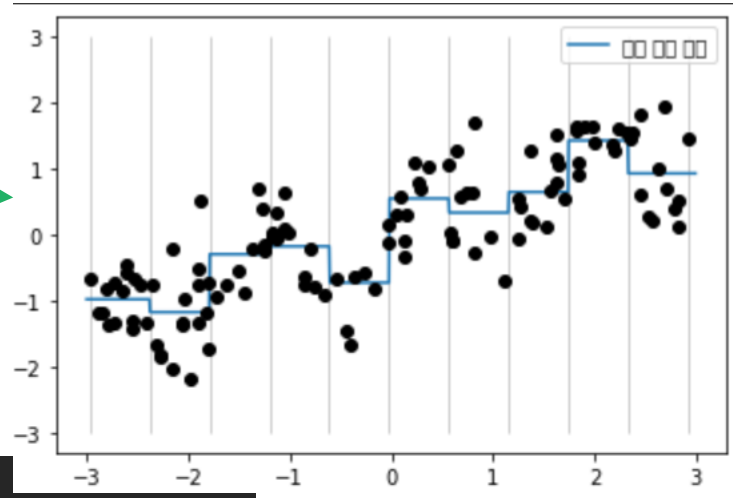
원본 데이터값:

```
[[-0.753]
 [ 2.704]
 [ 1.392]
 [ 0.592]
 [-2.064]
 [-2.064]
 [-2.651]
 [ 2.197]
 [ 0.607]
 [ 1.248]]
```

이산화된 데이터값:

```
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```

선형모델에서의 구간분할(이산화)



◆ 결과해석

구간분할한 선형모델이 트리모델과 같은 예측을 만들어냄

```
line = np.linspace(-3, 3, 1000, endpoint=False).reshape(-1, 1)

line_binned = kb.transform(line)
reg = LinearRegression().fit(X_binned, y)
plt.plot(line, reg.predict(line_binned))
plt.plot(X[:, 0], y, 'o', c='k')
plt.vlines(kb.bin_edges_[0], -3, 3, linewidth=1, alpha=.2)
plt.legend(loc="best")
plt.show()
```

상호작용과 다항식

◆ 특성을 조합해 더 좋은 모델 찾기

생성된 특성간의 상호작용 및 다항식을 추가하면 더 유용한 특성을 찾을 수 있다.

1. 원본 특성을 더한 선형회귀

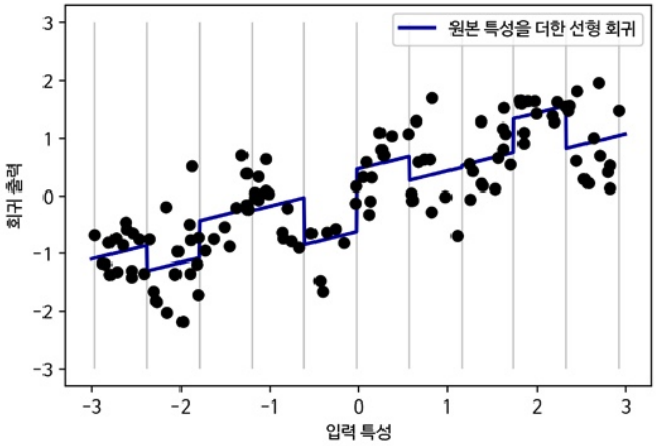
원본 특성 X, 이산화된 특성 X_binned (10개) -> 총 11개의 특성 -> 회귀모델 학습

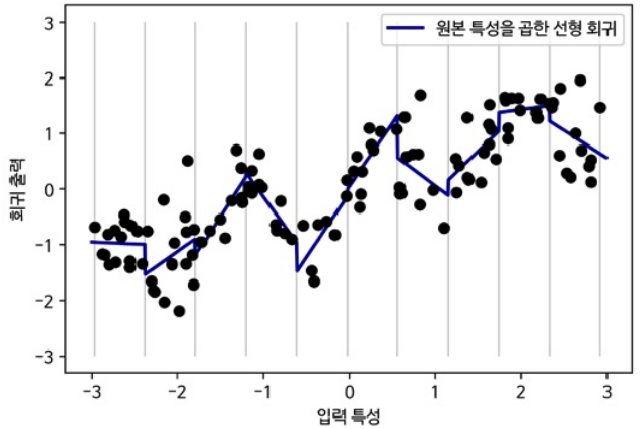
```
X_combined = np.hstack([X, X_binned])
reg = LinearRegression().fit(X_combined, y)
```

2. 원본 특성을 곱한 선형회귀

이산화된 특성 X_binned (10개) , 원본특성 X* X_binned (10개) -> 총 20개의 특성 -> 회귀모델 학습

```
X_product = np.hstack([X_binned, X * X_binned])
reg = LinearRegression().fit(X_product, y)
```



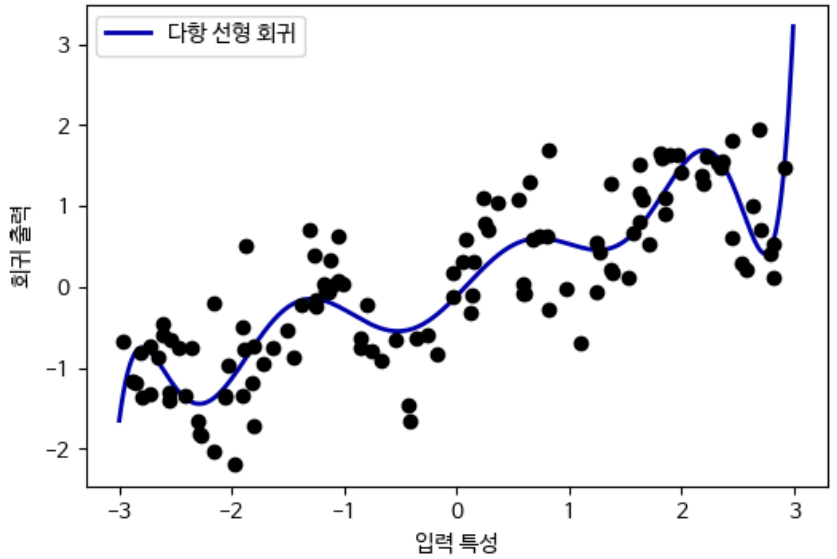


상호작용과 다항식

◆ 특성을 조합해 더 좋은 모델 찾기

3. 원본 특성에 다항식을 추가하기

원본 특성 X 에 대해 X의 2, 3, 4제곱을 특성으로 추가 -> 다항회귀식




```
from sklearn.preprocessing import PolynomialFeatures

# x ** 10까지 고차항을 추가
# 기본값인 "include_bias=True"는 절편에 해당하는 1인 특성을 추가
poly = PolynomialFeatures(degree=10, include_bias=False)
poly.fit(X)
X_poly = poly.transform(X)

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

일변량 비선형 변환

◆ Log, exp, sin, cos 등 수학 함수를 적용한 회귀모델

Input – target 이 비선형성을 띄는 데이터의 경우 수학 함수를 적용해 더 데이터에 맞는 선형회귀 모델을 만들 수 있음

Log, exp 함수: 데이터의 스케일을 변경
Sin, cos 함수: 주기적인 패턴이 있는 데이터에 잘 맞음

03. 특성선택 전략

특성선택의 배경

특성이 과도하게 추가되면 데이터 차원이 원본 특성의 수 이상으로 증가

- ▶ 모델 복잡도, 과대적합의 문제가 생김

따라서 생성된 특성 중 가장 유용한 특성만 선택해 모델에 적용할 필요가 있음

◆ 특성선택의 전략

자동적으로 가장 유용한 특성을 선택하는 알고리즘으로 변수선택법(variable selection)이라고도 부른다.

◆ 전략의 종류

일반적으로 가장 많이 쓰이는 것은 반복적 특성 선택 기법에 속하는 전진선택법, 후진선택법

그 외에도 일변량 통계 기법, 모델 기반 특성 선택법이 있음

특성 선택 전략의 종류

◆ 일변량 통계

분산분석, 개개의 특성과 target 사이에 중요한 통계적 관련이 있는 특성을 선택하는 방법

Point: 일변량, 즉 각 특성이 독립적으로 평가되기 때문에 다른 특성과 깊게 연관된 특성은 선택되지 않음

알고리즘: 분류 모델에서는 f_classif, 회귀모델에서는 f_regression을 보통 선택해 훈련 후 계산된 p-값에 기초해 특성을 제외함

라이브러리: SelectPercentile : 주어진 특성 중 지정된 비율만큼 특성을 선택

SelectKBest: 고정된 k개의 특성 선택

장점: 빠른 계산, 평가를 위한 모델을 만들지 않아도 됨, 특성 선택 후 적용할 모델에 상관없이 사용할 수 있음

```
from sklearn.feature_selection import SelectPercentile, f_classif

X_train, X_test, y_train, y_test = train_test_split(
    input, target, random_state=0, test_size=.5)
# 특성선택                    # score_fun: 특성을 선택하는 기준          # percentile : 비율을 지정하는 옵션
select = SelectPercentile(score_func=f_classif, percentile=50)
select.fit(X_train, y_train)
# 훈련 세트 적용
X_train_selected = select.transform(X_train)
```

특성 선택 전략의 종류

◆ 모델 기반 특성 선택

지도학습 머신러닝 모델을 사용해 가장 중요한 특성들만 선택하는 방법

(특성 선택에 사용하는 지도 학습 모델을 최종적으로 사용하지 않아도 됨)

Point: 한번에 모든 특성을 고려하므로 상호작용 부분을 반영할 수 있음

라이브러리: SelectFromModel : 중요도가 지정한 임계치보다 큰 모든 특성을 선택

특징 : 일변량 분석보다 훨씬 성능이 좋지만 복잡한 모델, 주로 결정 트리 모델의 특성 선택에 쓰임

```
from sklearn.feature_selection import SelectFromModel

# 특성선택
select = SelectFromModel(
    RandomForestClassifier(n_estimators=100, random_state=42),
    threshold="median")  # threshold : 임계치 지정
# 훈련 세트 적용
select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
```

03. 특성선택 전략

전진선택법, 후진소거법, 단계적 선택법

전진선택법(forward selection)

특성을 하나도 선택하지 않은 상태로 시작해서 특성 개수를 추가해가며 성능지표를 비교하는 방법

후진소거법(backward elimination)

모든 특성을 가지고 시작해 특성 개수를 제거해가며 성능지표를 비교해가는 방법, 후진소거법이라고도 불림

단계적 선택법(stepwise selection)

전진 선택법에 후진 소거법이 추가된 기법

전진선택법의 알고리즘

S : 기존 모형에 포함된 변수(특성)들의 집합

$\tilde{S}$: 모형에 포함되지 않은 변수들의 집합

α : 유의수준

집한 A에 대해 :

SSR : 집합 A의 모든 변수를 포함한 선형 모형의 회귀제곱합

SSE : 집합 A의 모든 변수를 포함한 선형 모형의 잔차제곱합

$card$: 집합 A의 변수 개수

Step1. 모델에 새 특성 추가하기

아직 모형에 포함되지 않은 변수 $X_k \in \tilde{S}$ 를 기존 모형에 추가하여 적합

Step2. 변수 X_k 에 대한 p-value 구하기

변수 X_k 에 대한 회귀계수 b_k 를 구하고 이에 대한 t통계량을 계산해 p-value를 구한다.

$$t = \left(\frac{SSR(X_k \cup S) - SSR(S)}{1} \div \frac{SSE(X_k \cup S)}{n - card(S) - 1} \right)^{\frac{1}{2}}$$

t통계량은 자유도가 n-card(S)-1인 t분포를 따르므로

$$p\text{-value} = P(T > t)$$

전진선택법의 알고리즘

Step3. 최소 p-value와 유의수준 α 비교하기

If p-value < α :

해당 변수 X_k 를 S 에 포함

else :

알고리즘 종료

후진소거법의 알고리즘

Step1.

현재 모형에 포함된 변수를 이용해 선형 모형에 적합

Step2.

추정된 회귀 계수에 대하여 가장 큰 p-value 값을 구함

Step3. 최소 p-value와 유의수준 α 비교하기

If p-value > α :

최대 p-value에 대응하는 변수를 기존 모형에서 제외

else :

알고리즘 종료

단계별 소거법의 알고리즘

Step1. 모델에 새 특성 추가하기

아직 모형에 포함되지 않은 변수 $X_k \in \tilde{S}$ 를 기존 모형에 추가하여 적합

Step2. 변수 X_k 에 대한 p-value 구하기

변수 X_k 에 대한 회귀계수 b_k 를 구하고 이에 대한 t통계량을 계산해 p-value를 구한다.

Step3. 최소 p-value와 유의수준 α 비교하기

If p-value < α :

해당 변수 X_k 를 S 에 포함시키고 Step 4로 넘어간다

else :

알고리즘 종료

단계별 소거법의 알고리즘

Step4. 추가된 변수를 포함해 선형 모형 적합

추가된 변수를 포함해 현재 모형에 있는 모든 변수를 사용해 선형 모형에 적합시키고 추정된 회귀 변수에 대하여 가장 큰 p-value값을 구함

Step5. 최대 p-value와 유의수준 α 비교하기

If p-value >= α :

해당 변수 X_k 를 모형에서 다시 제외하고 step1로 돌아감

else :

제외하는 변수 없이 step1로 돌아감

04. 코드 실습