Week8_회귀(5.1~5.8)

회귀 (Regression)

- 데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법
- **여러 개의 독립변수와 한 개의 종속변수 간의 상관관계**를 모델링하는 기법의 통칭

독립변수 = 피처

종속변수 = 결정 값 (label). 연속형 숫자 값으로 나옴

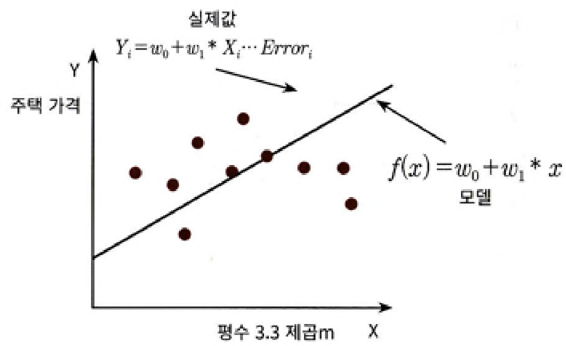
회귀 계수(Regression codfficients): 독립변수에 곱해지는 가중치

⇒ 머신러닝 회귀 예측에서 주어진 피처와 결정값 데이터 기반에서 학습을 통해 최적의 회귀 계수를 찾는 것이 중요함

회귀 유형 구분

독립변수 개수	회귀 계수의 결합
1개: 단일 회귀	선형: 선형 회귀
여러 개: 다중 회귀	비선형: 비선형 회귀

- 선형 회귀
  : 실제 값과 예측값의 차이 (오류의 제곱 값) 를 최소화하는 직선형 회귀선을 최적화하는 방식



** 규제 (Regularization): 선형 회귀의 과적합 문제를 해결하기 위해 회귀 계수에 패널티 값을 적용해 학습 데이터에 대한 정확도를 낮추는 것

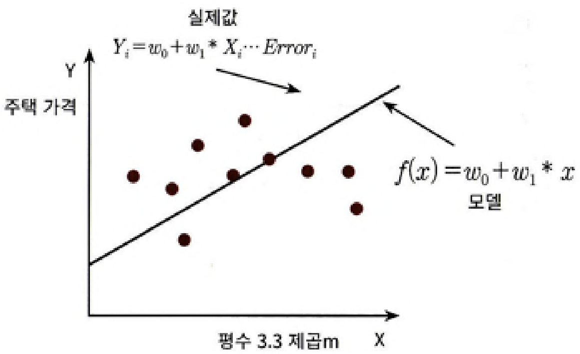
- 일반 선형 회귀: 예측값과 실제 값의 RSS(Resicual Sum of Squares)를 최소화할 수 있도록 회귀 계수를 최적화. 규제 적용 X
- 릿지(Ridge): 선형 회귀 + L2 규제
  - L2 규제: 상대적으로 더 큰 회귀 계수 값의 영향도를 감소시키기 위해 회귀 계수값을 더 작게 만듦
- 라쏘(Lasso): 선형 회귀 + L1 규제

- L1 규제 (피처 선택 기능): 예측 영향력이 작은 피처 회귀 계수를 0으로 만들어 예측 시 피처가 선택되지 않도록 함

  - 엘라스틱넷(ElasticNet): L2 + L1
  - 로지스틱 회귀(Logistic Regression): 분류에 사용되는 선형 모델

단순 선형 회귀

: 독립변수도 하나, 종속변수도 하나인 선형 회귀

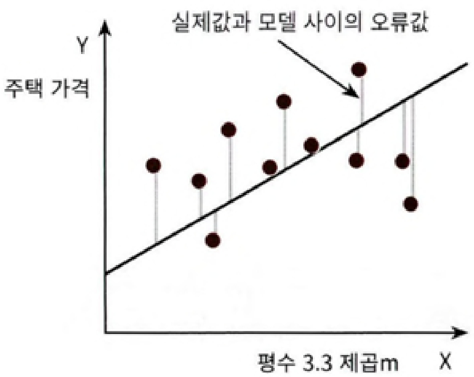


평수에 따라 증가하는 주택 가격에 대한 선형 모델 → 일차함수로 모델링됨

- 회귀 모델: $\hat{Y} = w_0 + w_1X$
- 실제 값: (1차 함수 값) - (오차)

오차 (잔차): 실제 값과 회귀 모델의 차이에 따른 오류 값

최적의 회귀 모델 만들기
= 전체 데이터의 잔차 합이 최소가 되는 모델 만들기
= 오류 값의 합이 최소가 될 수 있는 최적의 회귀 계수 찾기



⚠️ 오류 값은 양수와 음수 둘 다 가능하기 때문에 단순히 더하면 안 됨

→ 절댓값 취해서 더하기(Mean Absolute Error), 오류 값 제곱을 더하기(RSS, Residual Sum of Square = $Error^2$)

$$RSS(w_0, w_1) = \frac{1}{N} \sum_{i=1}^{N} (y_i - (w_0 + w_1 * x_i))^2$$

(i는 1부터 학습 데이터의 총 건수 N까지)

RSS는 **w 변수(회귀 계수)**가 중심 변수이며,
RSS를 최소로 하는 회귀 계수를 학습을 통해서 찾는 것이 ML 기반 회귀의 핵심!

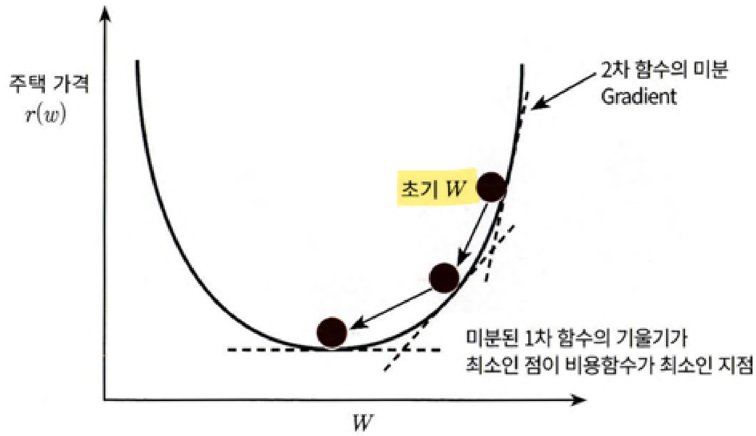
- RSS = 비용(Cost)
- w로 구성되는 RSS = 비용 함수 = 손실 함수 (loss function)

경사 하강법 (Gradient Descent)

❓ 어떻게 비용 함수가 최소 되는 w 파라미터를 구할 수 있는가?

경사 하강법

: 반복적으로 **RSS (오류 값) 가 작아지는 방향성**을 가지도록 w 파라미터를 지속해서 보정해 나가는 것. RSS이 더이상 작아지지 않으면 이 값을 최소 비용으로, 이때 w 값을 최적 파라미터로 반환한다.



⇒ 손실 함수가 2차 함수인 예시

(선형 회귀를 가정했을 때, 일차 함수를 제곱한 함수가 RSS이므로 이차 함수가 됨)

: 경사하강법은 최초 w에서 미분 적용해 미분 값이 계속 감소하는 방향으로 순차적으로 w를 업데이트
→ 더이상 미분 값이 감소하지 않는 지점 = 비용 함수가 최소인 지점
→ 이때의 w를 반환 (최적 파라미터)

경사 하강법 수식

⇒ R(w)의 미분 함수가 최솟값을 갖는 w 구하기

1. w1, w0을 임의의 값으로 설정하고 첫 비용 함수 값 계산
2. w1 = $w1 + \alpha * (2/N) * \sum_{i=1}^{N} x_i *$ (실제값i - 예측값i)
   w0 =
   $w0 + \alpha * (2/N) * \sum_{i=1}^{N}$(실제값i - 예측값i)
   업데이트한 뒤 비용 함수 값 다시 계산
3. 비용 함수가 감소하는 방향적으로 주어진 횟수만큼 2 반복하며 w1, w0 업데이트

▼ 설명

비용 함수 $RSS(w_0, w_1) = R(w) = \sum_{i=1}^{N}(y_i - (w_0 + w_1 * x_i))^2/N$

R(w)의 미분 함수의 최솟값을 구해야 하는데, w가 2개이므로 w0, w1에 대해 편미분한다.

R(w)를 w1, w0로 편미분한 결과

$$\frac{\partial R(w)}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} -x_t * (y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$\frac{\partial R(w)}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} (\text{실제값}_i - \text{예측값}_i)$$

보정 계수='**학습률**'을 적용

w1, w0을 (이전 w)+(학습률)*(편미분 값) 으로 반복적으로 업데이트

확률적 경사 하강법 (Stochastic Gradient Descent)

경사 하강법은 모든 학습 데이터에 대해 반복적으로 비용함수 최소화를 위한 값을 업데이트하기 때문에 수행 시간이 매우 오래 걸린다는 단점이 있음

→ 실전에서는 주로 (미니 배치) ***확률적 경사 하강법*** *이용*

: **일부 데이터만** 이용해 w가 업데이트되는 값 계산 → 빠른 속도

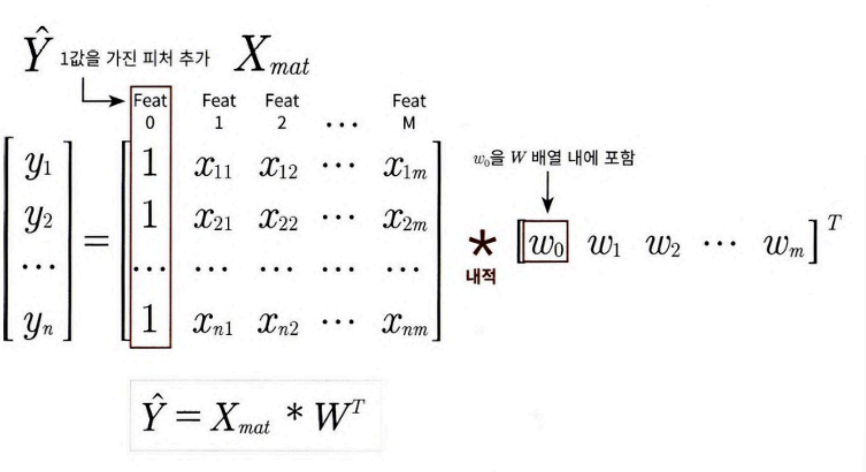
피처가 여러 개일 때 회귀 계수 도출하기

피처 M개 (X1, X2, ..., X100) → 회귀 계수 M+1 개 도출

$$\hat{Y} = w_0 + w_1 * X_1 + w_2 * X_2 + \cdots + w_{100} * X_{100}$$

데이터의 개수가 N, 피처 M개인 입력 행렬을 X_{mat}

회귀 계수 w1, w2, ..., w100 → W 배열



사이킷런 LinearRegression을 이용해 보스턴 주택 가격 예측하기

사이킷런 **linear_models** 모듈 → 다양한 선형 기반 회귀를 클래스로 구현해 제공

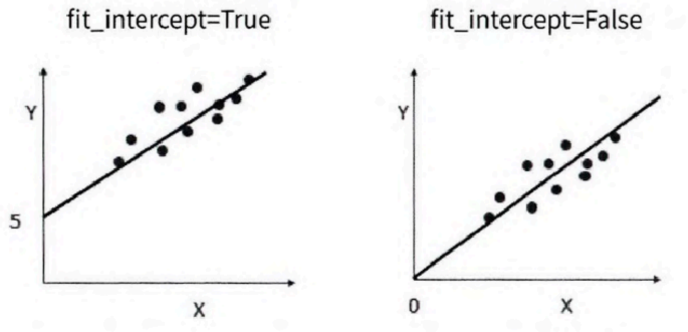
LinearRegression 클래스 - Ordinary Least Squares

: 규제가 적용되지 않은 선형 회귀를 구현한 클래스
: RSS 를 최소화해 OLS (Ordinary Least Squares) 추정 방식으로 구현한 클래스

fit() 메소드로 X, y 배열을 입력받으면 회귀 계수 W를 coef_ 속성에 저장

```
class sklearn.linear_model.LinearRegression(fit_intercept=True,
                                            normalize=False, copy_X=True, n_jobs=1)
```

파라미터	fit_intercept	- boolean 값 - default=True - intercept(절편) 값을 계산할지 말지 지정. False로 지정하는 경우 0으로 지정됨
	normalize	- boolean 값 - default=False - 회귀를 수행하기 전에 입력 데이터 세트를 정규화할지 말지 지정
속성	coef_	- fit() 메소드를 수행했을 때 회귀 계수가 배열 형태로 저장하는 속성 - Shape = (Target 값 개수, 피처 개수)
	intercept_	- intercept 값



✅ OLS 기반 회귀 계수 계산은 입력 피처의 독립성에 많은 영향을 받음

다중 공선성 (multi-collinearity) 문제

: 피처 간의 상관관계가 매우 높아 분산이 커져서 오류에 민감해지는 현상

⇒ 상관관계가 높은 피처가 많은 경우
독립적인 중요한 피처만 남기고 제거하거나 규제 적용

⇒ 매우 많은 피처가 다중 공선성 문제를 가지고 있는 경우
PCA를 통해 차원 축소 수행

회귀 평가 지표

실제 값과 회귀 예측값의 차이 값을 기반으로 한 지표가 중심

평가 지표	설명	수식
MAE	Mean Absolute Error(MAE)이며 실제 값과 예측값의 차이를 절댓값으로 변환해 평균한 것입니다.	$MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert Yi - \hat{Y}i \rvert$
MSE	Mean Squared Error(MSE)이며 실제 값과 예측값의 차이를 제곱해 평균한 것입니다.	$MSE = \frac{1}{n}\sum_{i=1}^{n}(Yi - \hat{Y}i)^2$
RMSE	MSE 값은 오류의 제곱을 구하므로 실제 오류 평균보다 더 커지는 특성이 있으므로 MSE에 루트를 씌운 것이 RMSE(Root Mean Squared Error)입니다.	$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Yi - \hat{Y}i)^2}$
R^2	분산 기반으로 예측 성능을 평가합니다. 실제 값의 분산 대비 예측값의 분산 비율을 지표로 하며, 1에 가까울수록 예측 정확도가 높습니다.	$R^2 = \frac{\text{예측값 } Variance}{\text{실제값 } Variance}$

+) MSLE (Mean Squared **Log** Error), RMSLE (Root Mean Squared **Log** Error)

각 평가 방법에 대한 사이킷런의 API 및 scoring 파라미터 적용 값

평가 방법	사이킷런 평가 지표 API	Scoring 함수 적용 값
MAE	metrics.mean_absolute_error	'neg_mean_absolute_error'
MSE	metrics.mean_squared_error	'neg_mean_squared_error'
RMSE	metrics.mean_squared_error를 그대로 사용하되 squared 파라미터를 False로 설정.	'neg_root_mean_squared_error'
MSLE	metrics.mean_squared_log_error	'neg_mean_squared_log_error'
R^2	metrics.r2_score	'r2'

❓ **neg_** 접두어 - Scoring 함수에 음수값을 반환하는 이유

: 사이킷런의 Scoring 함수가 score 값이 클수록 좋은 평가 결과로 자동 평가하기 때문에, 손실 값이 최소가 되도록 하는 회귀 평가 지표에 적용하기 위해서는 음수값으로 보정해야 함

다항 회귀와 과(대)적합/과소적합 이해

다항 회귀 (Polynomial Regression)

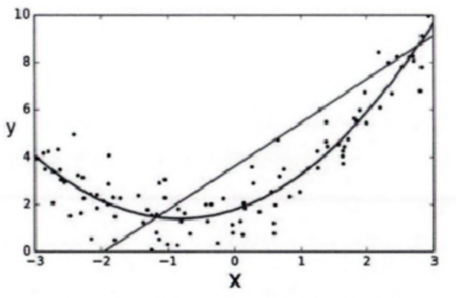
: 회귀가 독립변수의 단항식이 아닌 2차, 3차 방정식과 같은 다항식으로 표현되는 것

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + x_5x_2^2$$

✅ 다항 회귀도 **선형 회귀**임

: 선형 회귀/비선형 회귀를 나누는 기준은 **회귀 계수가 선형/비선형인지**에 따른 것

위의 식에서 새로운 변수 $z = [x_1, x_2, x_1x_2, x_1^2, x_2^2]$ 로 두면 여전히 선형 회귀!



〈 주어진 데이터 세트에서 다항 회귀가 더 효과적임 〉

사이킷런에서 다항 회귀를 위한 클래스를 명시적으로 제공하진 않지만, 다항 회귀 또한 선형 회귀라는 점을 이용해 비선형 함수를 선형 모델에 적용시킬 수 있다.

→ **PolynomialFeatures** 클래스를 통해 피처를 Polynomial(다항식) 피처로 변환

PolynomialFeatures 클래스

: degree 파라미터를 통해 입력받은 단항식 피처를 degree에 해당하는 다항식 피처로 변환

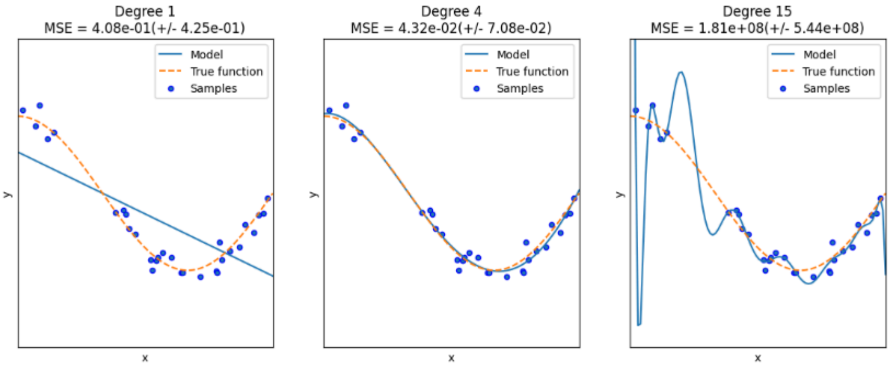
— fit(), transform() 으로 수행

다항 회귀를 이용한 과소적합 및 과적합 이해

다항 회귀는 직선 관계가 아닌 복잡한 다항 관계를 모델링할 수 있고 다항식의 차수가 높아질수록 (그래프가 꼬불꼬불해짐) 매우 복잡한 피처 간의 관계까지 모델링 가능함.

⚠️ 그러나 **다항 회귀의 차수(degree)를 높일수록 과적합 문제** 발생

편향-분산 트레이드오프

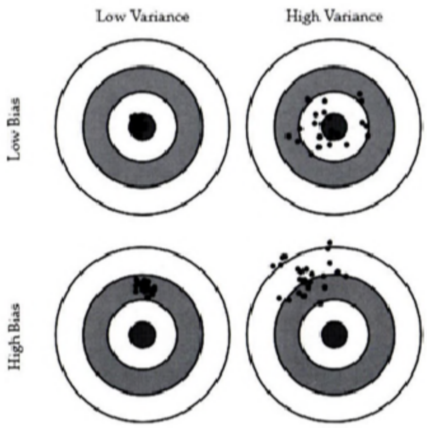


Degree 1 모델: 매우 단순화된 모델, 지나치게 한 방향성으로 치우친 경향

→ 고편향 (High Bias)

Degree 15 모델: 매우 복잡한 모델, 지나치게 높은 변동성

→ 고분산 (High Variance)

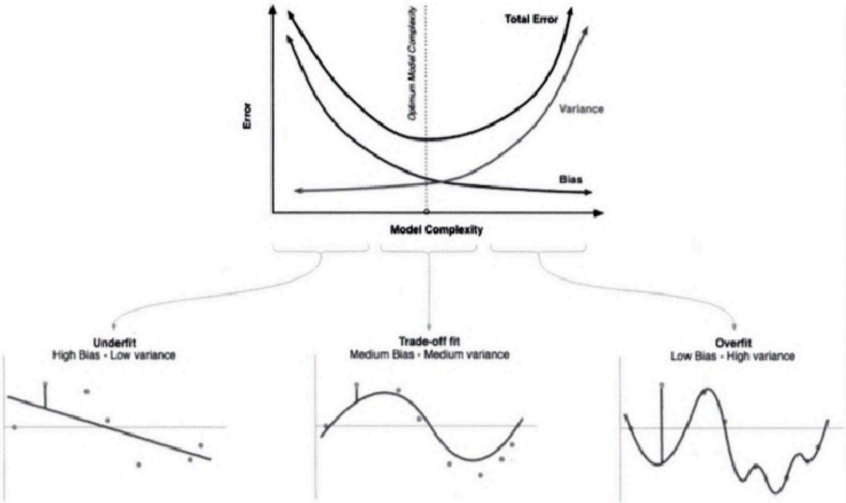


〈 편향과 분산의 고/저에 따른 표현.
http://scott.fortmann-roe.com/docs/BiasVariance.html에서 발췌 〉

편향-분산 트레이드오프

: 일반적으로 편향과 분산은 한쪽이 높으면 한쪽이 낮아짐.

- 고편향 저분산 ⇒ 과소적합
- 저편향 고분산 ⇒ 과적합



〈 편향과 분산에 따른 전체 오류 값(Total Error) 곡선. http://scott.fortmann-roe.com/docs/BiasVariance.html에서 발췌. 〉

편향이 너무 높을 때 → 전체 오류가 높음

⇒ 편향을 점점 낮추면 → 분산이 높아지고 전체 오류도 낮아짐

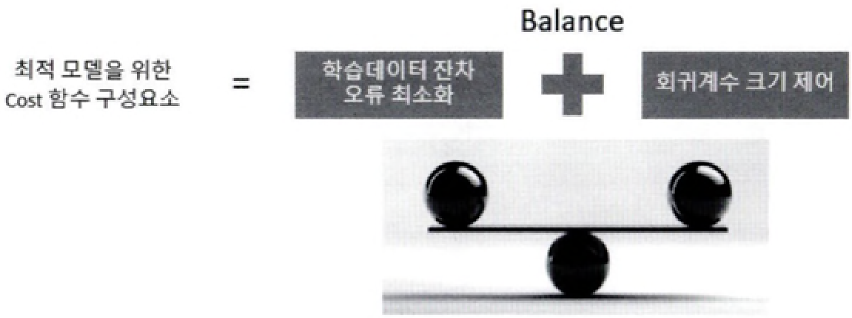
⇒ 전체 오류가 가장 낮아지는 **골디락스** 지점

⇒ 분산을 계속 높이면 → 전체 오류 값이 다시 증가함. 예측 성능 저하

규제 선형 모델 - 릿지, 라쏘, 엘라스틱넷

규제 선형 모델

⚠️ 회귀 모델은 적절히 데이터에 적합하면서도 회귀 계수가 기하급수적으로 커지는 것을 제어할 수 있어야 함



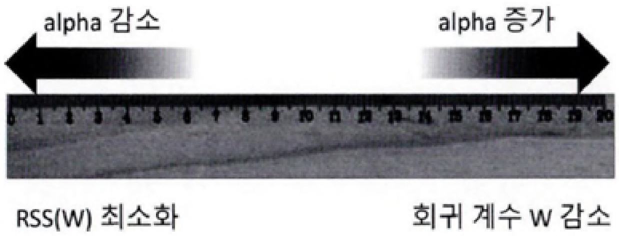
회귀 계수의 크기를 제어해 과적합을 개선하려면?

$$\text{비용 함수 목표} = \mathrm{Min}(\mathrm{RSS(W)} + \mathrm{alpha} * \|W\|_2^2)$$

alpha: 학습 데이터 적합도와 회귀 계수 값의 크기 제어를 수행하는 튜닝 파라미터

- alpha = 0 → W가 커도 **비용 함수 목표 = Min(RSS(W))**

- alpha = 무한대 → RSS(W)에 비해 $alpha * ||W||_2^2$ 가 너무 커지므로, **W 값을 0에 가깝게 작게 만들어야** Cost가 최소화되는 비용 함수 목표를 달성할 수 있음



〈alpha 튜닝 파라미터를 통한 RSS 최소화와 회귀 계수 크기 감소의 균형 조정〉

규제 (Regularization)

: 비용 함수에 alpha 값으로 페널티를 부여해 회귀 계수 값의 크기를 감소시켜 과적합을 개선하는 방식

- L2 규제 = W의 제곱에 대해 페널티를 부여하는 방식 → **릿지 (Ridge) 회귀**
- L1 규제 = W의 절댓값에 대해 페널티를 부여하는 방식 → **라쏘 (Lasso) 회귀**

릿지 회귀

L2 규제 = W의 제곱에 대해 페널티를 부여하는 방식

사이킷런은 **Ridge 클래스**로 릿지 회귀 구현

- alpha = 릿지 회귀의 alpha L2 규제 계수

	alpha:0	alpha:0.1	alpha:1	alpha:10	alpha:100
RM	3.809865	3.818233	3.854000	3.702272	2.334536
CHAS	2.686734	2.670019	2.552393	1.952021	0.638335
RAD	0.306049	0.303515	0.290142	0.279596	0.315358
ZN	0.046420	0.046572	0.047443	0.049579	0.054496
INDUS	0.020559	0.015999	-0.008805	-0.042962	-0.052826
B	0.009312	0.009368	0.009673	0.010037	0.009393
AGE	0.000692	-0.000269	-0.005415	-0.010707	0.001212
TAX	-0.012335	-0.012421	-0.012912	-0.013993	-0.015856
CRIM	-0.108011	-0.107474	-0.104595	-0.101435	-0.102202
LSTAT	-0.524758	-0.525966	-0.533343	-0.559366	-0.660764
PTRATIO	-0.952747	-0.940759	-0.876074	-0.797945	-0.829218
DIS	-1.475567	-1.459626	-1.372654	-1.248808	-1.153390
NOX	-17.766611	-16.684645	-10.777015	-2.371619	-0.262847

alpha 값이 커짐에 따라 회귀 계수들이 작아지는 것을 확인

라쏘 회귀

L1 규제 = W의 절댓값에 대해 페널티를 부여하는 방식

라쏘 회귀 비용함수의 목표 = $RSS(W) + alpha * ||W||_1$ 식을 최소화하는 W 찾기

- L1 규제: **불필요한 회귀 계수를 급격히 감소시켜 0으로 만들어 제거**

  → 적절한 피처만 회귀에 포함시키는 **피처 선택**의 특성
- L2 규제: 회귀 계수의 크기 감소

사이킷런 **Lasso 클래스**를 통해 라쏘 회귀 구현

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
RM	3.789725	3.703202	2.498212	0.949811	0.000000
CHAS	1.434343	0.955190	0.000000	0.000000	0.000000
RAD	0.270936	0.274707	0.277451	0.264206	0.061864
ZN	0.049059	0.049211	0.049544	0.049165	0.037231
B	0.010248	0.010249	0.009469	0.008247	0.006510
NOX	-0.000000	-0.000000	-0.000000	-0.000000	0.000000
AGE	-0.011706	-0.010037	0.003604	0.020910	0.042495
TAX	-0.014290	-0.014570	-0.015442	-0.015212	-0.008602
INDUS	-0.042120	-0.036619	-0.005253	-0.000000	-0.000000
CRIM	-0.098193	-0.097894	-0.083289	-0.063437	-0.000000
LSTAT	-0.560431	-0.568769	-0.656290	-0.761115	-0.807679
PTRATIO	-0.765107	-0.770654	-0.758752	-0.722966	-0.265072
DIS	-1.176583	-1.160538	-0.936605	-0.668790	-0.000000

alpha 크기가 증가함에 따라 일부 피처의 회귀 계수는 아예 0으로 바뀌고 있음

-> 피처 선택의 효과

엘라스틱넷 회귀

= L2 규제 + L1 규제

$$\mathrm{RSS(W)} + \mathrm{alpha2} * \|W\|_2^2 + \mathrm{alpha1} * \|W\|_1$$

위의 식을 최소화하는 W를 찾는 것이 목표

- 장점: 라쏘 회귀의 변동성 완화
  - 라쏘 회귀(L1)는 서로 상관관계가 높은 피처들의 경우 이들 중 중요 피처만 고르고 다른 피처들의 회귀 계수는 전부 0으로 만들어버림

    → alpha 값에 따라 회귀 계수의 값이 급격히 변동할 수 있음

    ⇒ L2 규제를 추가해 이를 완화
- 단점: 수행 시간 ⬆️

사이킷런 **ElasticNet 클래스**를 통해 엘라스틱넷 회귀 구현

- 파라미터
  - alpha = a * L1 + b * L2 에서 a + b
  - l1_ratio = a / (a + b). L1 규제 적용 비율

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
RM	3.574162	3.414154	1.918419	0.938789	0.000000
CHAS	1.330724	0.979706	0.000000	0.000000	0.000000
RAD	0.278880	0.283443	0.300761	0.289299	0.146846
ZN	0.050107	0.050617	0.052878	0.052136	0.038268
B	0.010122	0.010067	0.009114	0.008320	0.007020
AGE	-0.010116	-0.008276	0.007760	0.020348	0.043446
TAX	-0.014522	-0.014814	-0.016046	-0.016218	-0.011417
INDUS	-0.044855	-0.042719	-0.023252	-0.000000	-0.000000
CRIM	-0.099468	-0.099213	-0.089070	-0.073577	-0.019058
NOX	-0.175072	-0.000000	-0.000000	-0.000000	-0.000000
LSTAT	-0.574822	-0.587702	-0.693861	-0.760457	-0.800368
PTRATIO	-0.779498	-0.784725	-0.790969	-0.738672	-0.423065
DIS	-1.189438	-1.173647	-0.975902	-0.725174	-0.031208

alpha = 0.5일 때 RMSE가 가장 좋은 예측 성능을 보임

alpha 값에 따른 피처들의 회귀 계수 값이 라쏘보다는 상대적으로 0이 되는 값이 적다

선형 회귀 모델을 위한 데이터 변환

→ 데이터 분포도 정규화 & 인코딩 방법 선정이 매우 중요!

✅ 중요 피처들이나 타깃값의 분포도가 심하게 왜곡된 경우, 데이터에 대한 스케일링/정규화(정규 분포) 작업을 수행

사이킷런을 이용해 피처 데이터 세트에 적용하는 **변환 작업**

1. StandardScaler 클래스 → 평균 0 분산 1인 표준 정규 분포를 가진 데이터 세트로 변환
   MinMaxScaler 클래스 → 최솟값 0 최댓값 1인 값으로 정규화
   ⚠️ 예측 성능 향상을 크게 기대하기 어려움
2. 스케일링/정규화를 수행한 데이터 세트에 다시 다항 특성을 적용하여 변환

   1번 수행 후 예측 성능이 향상되지 않을 경우 이와 같은 방법 적용
   ⚠️
   피처 개수가 매우 많을 때는 다항 변환으로 생성되는 피처 개수가 기하급수적으로 늘어남, 과적합 이슈

3. **로그 변환 (Log Transformation)** → 보다 정규 분포에 가까운 형태로 값이 분포됨
   ✔ 많이 사용되는 방법!

** 타겟 값도 일반적으로 로그 변환을 적용함. 결정 값을 정규 분포 등으로 변환하면 변환된 값을 다시 원본 타겟 값으로 원복하기 어려울 수 있기 때문.

변환 유형	alpha 값			
	alpha=0.1	alpha=1	alpha=10	alpha=100
원본 데이터	5.788	5.653	5.518	5.330
표준 정규 분포	5.826	5.803	5.637	5.421
표준 정규 분포 + 2차 다항식	8.827	6.871	5.485	**4.634**
최솟값/최댓값 정규화	5.764	5.465	5.754	7.635
최솟값/최댓값 정규화 + 2차 다항식	5.298	**4.323**	5.185	6.538
로그 변환	**4.770**	**4.676**	**4.836**	6.241

→ 표준 정규 분포 & 최솟값/최댓값 정규화: 성능상 개선 X

→ 표준 정규 분포 + 2차 다항식: alpha=100에서 4.634로 개선됨

→ 최솟값/최댓값 정규화 + 2차 다항식: alpha=1에서 4.323으로 개선됨

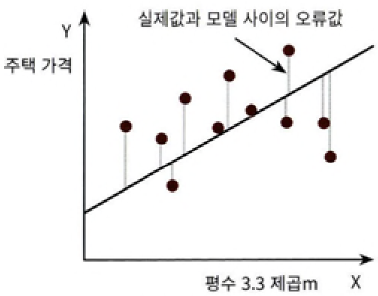
그러나 다항식 변환은 피처 개수가 많으면 적용하기 힘듦, 데이터 건수가 많아지면 계산에 많은 시간이 소모되어 적용에 한계가 있음

→ 로그 변환: alpha=0.1, 1, 10인 경우 모두 좋은 성능 향상 O

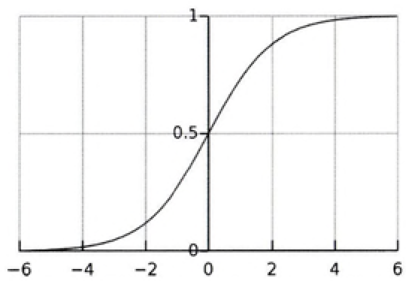
로지스틱 회귀

: 선형 회귀 방식을 분류에 적용한 알고리즘

✅ **시그모이드 (Sigmoid) 함수** 최적선을 찾고, 시그모이드 함수의 반환 값을 확률로 간주해 확률에 따라 분류를 결정



〈 선형 회귀의 선형 함수 〉

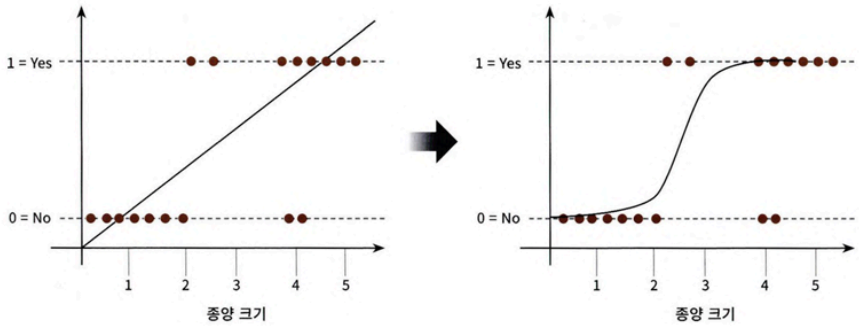


〈 로지스틱 회귀의 시그모이드 함수 〉

$$y = \frac{1}{1+e^{-x}}$$

시그모이드 함수의 정의

- y 값이 항상 0과 1 사이 값
  - x 값 증가 ⇒ y → 1
  - x 값 감소 ⇒ y → 0
  - x == 0 ⇒ y = 0.5
- S자 커브 형태 → 선형 그래프보다 좀 더 정확하게 0과 1에 대해 분류해줌



사이킷런 **LogisticRegression 클래스** 제공

- 클래스의 **solver** 파라미터의 ‘lbfgs’, ‘liblinear’, ‘newton-cg’, ‘sag’, ‘saga’ 값을 적용해 최적화 선택 가능

  - lbfgs: 사이킷런 버전 0.22부터 solver의 기본 설정값입니다. 메모리 공간을 절약할 수 있고, CPU 코어 수가 많다면 최적화를 병렬로 수행할 수 있습니다.
  - liblinear: 사이킷런 버전 0.21까지에서 solver의 기본 설정값입니다. 다차원이고 작은 데이터 세트에서 효과적으로 동작하지만 국소 최적화(Local Minimum)에 이슈가 있고, 병렬로 최적화할 수 없습니다.
  - newton-cg: 좀 더 정교한 최적화를 가능하게 하지만, 대용량의 데이터에서 속도가 많이 느려집니다.
  - sag: Stochastic Average Gradient로서 경사 하강법 기반의 최적화를 적용합니다. 대용량의 데이터에서 빠르게 최적화합니다.
  - saga: sag와 유사한 최적화 방식이며 L1 정규화를 가능하게 해줍니다.

  → 성능 차이는 미비하고 일반적으로 lbfgs 또는 liblinear 선택

- **penalty:** 규제 유형 설정 - ‘l2’(default), ‘l1’
- **C:** 1 / alpha. C가 클수록 규제 강도 커짐

회귀 트리

- 선형 회귀
  :

회귀 계수를 선형으로 결합하는 회귀 함수를 구해, 독립 변수를 입력해 결괏값을 예측

- **비선형 회귀**
  :
  회귀 계수의 결합이 비선형인 비선형 회귀 함수를 통해 결괏값을 예측

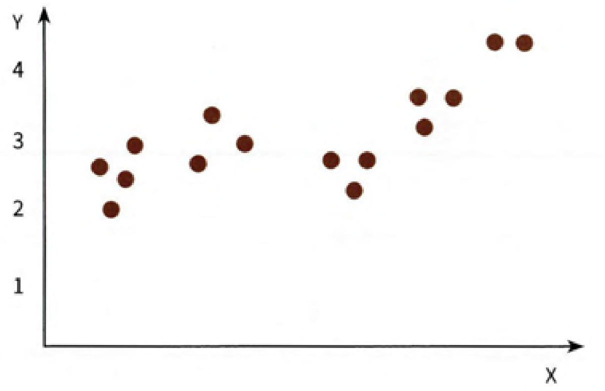
💡 회귀 트리

= 트리 기반의 회귀에서 사용하는 트리

= 회귀를 위한 트리를 생성하고 이를 기반으로 회귀 예측

→ **리프 노드에 속한 데이터 값의 평균값**을 구해 회귀 예측값 계산

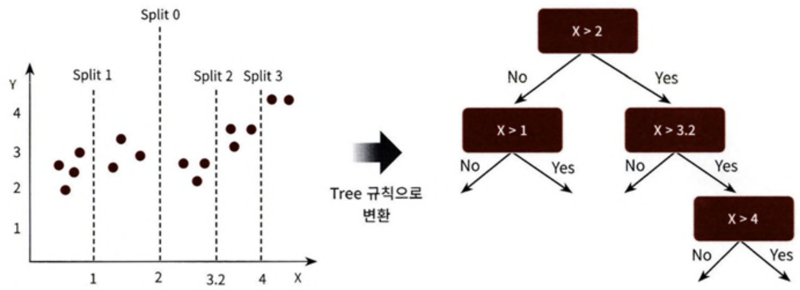
회귀 트리 동작 원리



피처가 하나인 X 피처 데이터 세트 & 결정값 Y

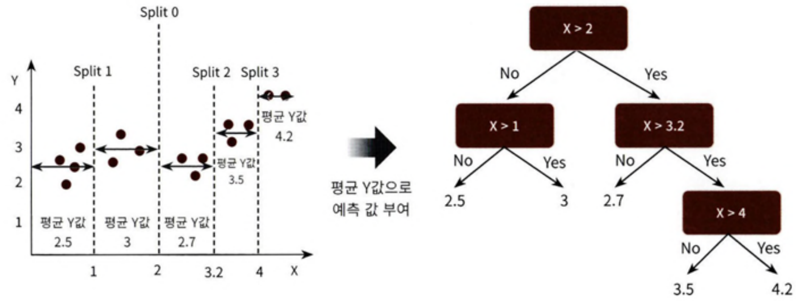
1. X 피처를 **결정 트리 기반으로 분할**

   → X 값의 균일도 반영한 지니 계수에 따라 다음과 같이 분할됨



2. 리프 노드 생성 기준에 부합하는 트리 분할 완료

   → **리프 노드에 소속된 데이터 값의 평균값**을 구해 최종적으로 **리프 노드에 결정 값**으로 할당



** 트리 생성이 CART(Classification And Regression Trees) 알고리즘에 기반하고 있기 때문에, 모든 트리 기반 알고리즘은 분류와 회귀 모두 가능

알고리즘	회귀 Estimator 클래스	분류 Estimator 클래스
Decision Tree	DecisionTreeRegressor	DecisionTreeClassifier
Gradient Boosting	GradientBoostingRegressor	GradientBoostingClassifier
XGBoost	XGBRegressor	XGBClassifier
LightGBM	LGBMRegressor	LGBMClassifier