

규제 선형회귀 입문

Ridge · Lasso · Elastic Net 원리와 적용

이 가이드는 선형회귀를 처음 접하는 분들도 따라올 수 있도록
원리와 R/Python 코드를 함께 다룹니다.

1. 들어가며

이 가이드는 회귀분석에서 가장 널리 쓰이는 규제(regularization) 기법 세 가지, Ridge 회귀, Lasso 회귀, Elastic Net 을 다룹니다. 세 방법 모두 “변수가 많거나 변수끼리 상관관계가 있을 때 일반적인 선형회귀(OLS)가 불안정해지는 문제”를 해결하기 위해 고안되었습니다.

선형회귀 자체가 낯선 분들도 따라올 수 있도록 2 장에서 기본 개념부터 짚고 시작합니다. 이미 익숙하다면 2 장은 건너뛰고 3 장부터 보아도 무방합니다.

예제 데이터셋

이 가이드는 처음부터 끝까지 하나의 데이터셋으로 세 방법을 일관되게 비교합니다. scikit-learn 에 내장된 Diabetes 데이터셋(당뇨병 환자 442 명의 나이, BMI, 혈압, 혈청 지표 등 10 개 변수로 1 년 후 질병 진행도를 예측)을 기본으로 사용하되, 다음을 추가했습니다.

- noise1, noise2, noise3: target 과 사실상 무관한 순수 잡음 변수 3 개. Lasso 와 Elastic Net 이 이런 변수들을 어떻게 속아내는지 보여주기 위해 추가했습니다.
- noise3 는 다른 변수들과 스케일을 의도적으로 다르게(표준편차 기준 약 240 배 크게) 설정했습니다. 이는 3 장에서 “왜 변수 표준화가 필요한가”를 보여주는 용도로 사용합니다.

원본 10 개 변수 중 s1(총 콜레스테롤)과 s2(LDL 콜레스테롤)는 상관관계수 0.90 으로 강하게 얽혀 있습니다. 이는 인위적으로 만든 것이 아니라 원본 데이터에 실제로 존재하는 다중공선성이며, 이 가이드 전체에서 Ridge 와 Lasso 의 차이를 보여주는 핵심 사례로 사용됩니다.

데이터 준비

Python 에서 데이터를 만들고 CSV 로 저장한 뒤, R 에서도 같은 CSV 를 불러와 사용합니다. 이렇게 해야 두 언어의 결과가 완전히 동일한 데이터를 기준으로 비교됩니다.

R

```
# Python 에서 미리 저장한 훈련 데이터(80%)를 읽어옵니다
df <- read.csv("diabetes_train.csv")
X <- df[, !(names(df) %in% c("target"))]
y <- df$target
```

Python

```
from sklearn.datasets import load_diabetes
import numpy as np, pandas as pd

np.random.seed(42)
data = load_diabetes(as_frame=True)
df = data.frame.copy()
n = len(df)

df['noise1'] = np.random.normal(0, 0.05, n)
df['noise2'] = np.random.normal(0, 0.05, n)
df['noise3'] = np.random.uniform(-20, 20, n) # 스케일을 의도적으로 깨뜨림

df.to_csv('diabetes_extended.csv', index=False)
from sklearn.model_selection import train_test_split
X = df.drop(columns='target'); y = df['target']
X_train, X_test, y_train, y_test = train_test_split(
```

```
X, y, test_size=0.2, random_state=42)
X_train.assign(target=y_train).to_csv('diabetes_train.csv', index=False)
X_test.assign(target=y_test).to_csv('diabetes_test.csv', index=False)
```

2. 선형회귀와 과적합

2-1. 선형회귀란 무엇인가

선형회귀는 입력 변수들(X)의 가중합으로 결과값(y)을 예측하는 가장 기본적인 모델입니다.

$$\hat{y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

여기서 β (베타)는 각 변수가 예측에 얼마나 기여하는지를 나타내는 계수입니다. 좋은 계수를 찾는 기준은 실제값과 예측값의 차이(잔차)를 제곱해 더한 값, 즉 잔차제곱합(RSS)을 최소화하는 것입니다.

$$RSS = \sum (y_i - \hat{y}_i)^2$$

이 RSS를 최소화하는 계수를 찾는 방법을 최소제곱법(OLS, Ordinary Least Squares)이라 부릅니다. 변수가 몇 개 안 되고 서로 독립적이라면 OLS는 안정적이고 해석하기 쉬운 결과를 줍니다.

2-2. 과적합이란 무엇인가

문제는 변수가 많아지거나, 변수끼리 서로 강하게 상관되어 있을 때 발생합니다. 모델이 데이터의 일반적인 패턴이 아니라 그 데이터에만 있는 우연한 잡음까지 외워버리는 현상을 과적합(overfitting)이라 부릅니다.

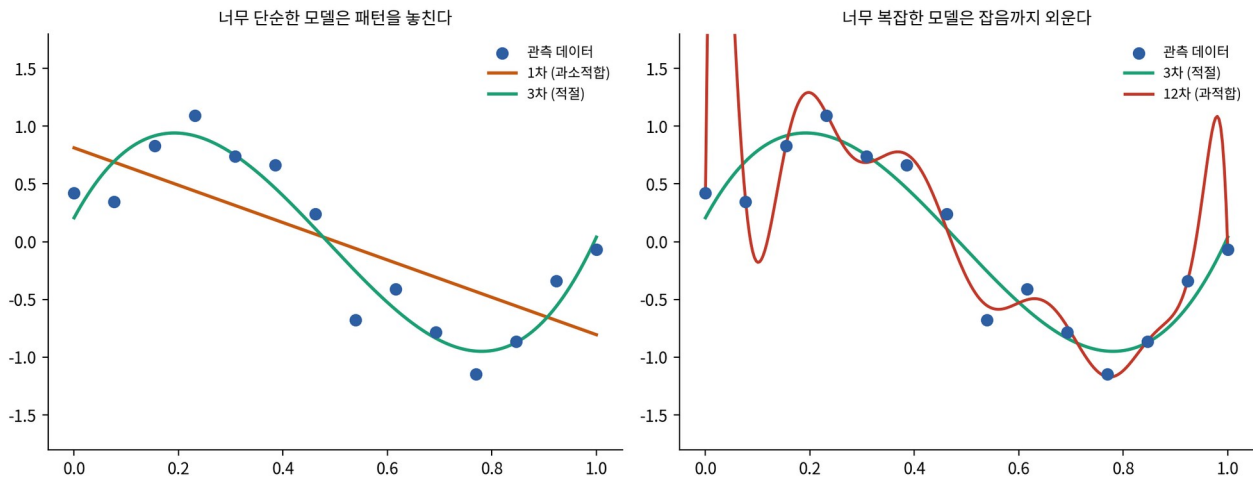


그림 1. 같은 데이터에 대해 차수가 다른 다항식을 적합한 예. 1차(왼쪽, 주황)는 패턴을 충분히 따라가지 못하고, 12차(오른쪽, 빨강)는 점들을 거의 정확히 통과하지만 점과 점 사이에서 데이터에 없던 요동을 만들어낸다. 3차(초록)가 가장 적절한 균형을 보여준다.

그림 1의 12차 곡선은 훈련 데이터에 대해서는 거의 완벽한 적합을 보이지만, 정작 새로운 데이터가 들어오면 엉뚱한 예측을 합니다. 회귀계수의 관점에서 보면, 과적합된 모델은 보통 계수의 절댓값이 비정상적으로 크고 부호가 불안정합니다 — 아주 작은 데이터 변화에도 곡선이 크게 출렁여야 하기 때문입니다.

다중공선성(변수끼리 강하게 상관된 상황)도 비슷한 문제를 일으킵니다. 두 변수가 거의 같은 정보를 담고 있으면, OLS는 그 둘에게 계수를 어떻게 나눠줄지 거의 무작위로 결정하게 되어 계수가 극단적으로 크거나 부호가 뒤바뀌는 등 불안정해집니다. 이 가이드의 예제 데이터셋에서 s1과 s2(상관계수 0.90)가 정확히 이런 상황을 보여줍니다 — 6장에서 실제 수치로 확인합니다.

2-3. “계수를 작게 누르면 어떨까?”

과적합된 모델의 계수가 비정상적으로 크다면, 거꾸로 계수가 너무 커지지 않도록 처음부터 제약을 거는 건 어떨까요? 이것이 규제(regularization)의 핵심 아이디어입니다.

구체적으로는, OLS 가 최소화하던 잔차제곱합(RSS)에 “계수가 클수록 불이익을 주는 항”을 더해서 함께 최소화합니다.

$$\text{목적함수} = \text{RSS} + \text{페널티(계수의 크기)}$$

이 페널티를 어떻게 정의하느냐에 따라 Ridge, Lasso, Elastic Net 이 갈라집니다. 다음 장부터 하나씩 살펴봅니다.

3. Ridge 회귀 (L2 규제)

시작하기 전에: 왜 변수 스케일링이 필요한가

페널티 항은 계수 β_j 들의 크기에 직접 작동합니다. 그런데 변수들의 측정 단위(스케일)가 서로 다르면 문제가 생깁니다. 예를 들어 한 변수는 0~1 범위이고 다른 변수는 0~1,000,000 범위라면, 같은 정도로 중요한 변수라도 스케일이 작은 쪽은 같은 영향력을 내기 위해 훨씬 큰 계수가 필요합니다. 페널티는 계수의 절대적 크기만 보고 작동하므로, 규제가 변수의 실제 중요도가 아니라 측정 단위에 의해 좌우되는 왜곡이 생깁니다.

그래서 Ridge, Lasso, Elastic Net 을 적용하기 전에는 반드시 모든 변수를 평균 0, 표준편차 1 로 표준화 (standardization)해야 합니다. 이 가이드의 noise3 변수로 이 왜곡을 직접 확인해 보겠습니다.

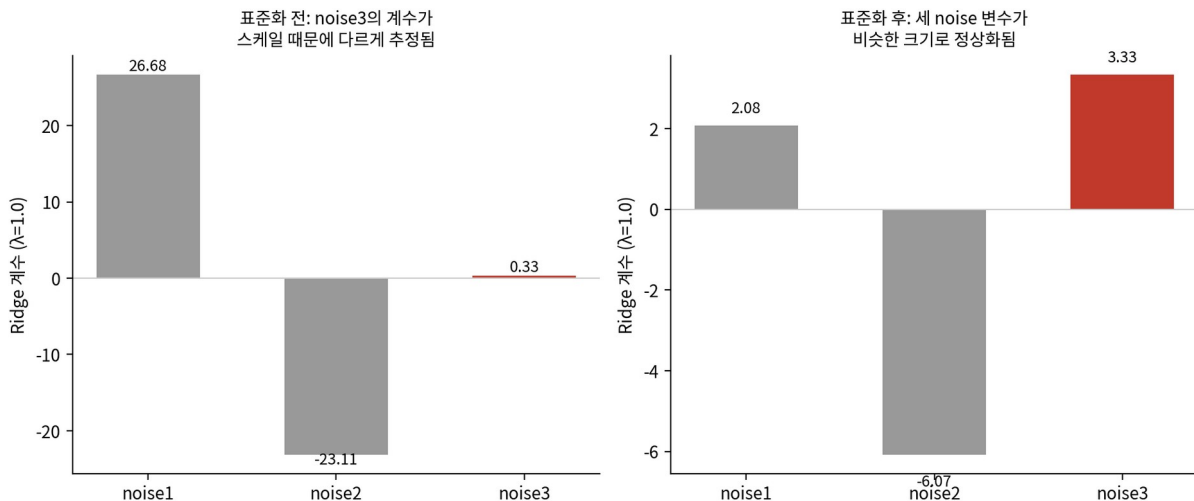


그림 2. 표준화 전(왼쪽)과 후(오른쪽)의 noise1·noise2·noise3 Ridge($\lambda=1$) 계수 비교. noise3(빨강)는 다른 두 변수보다 표준편차가 약 240 배 크게 설정되어 있다. 표준화 전 noise3의 계수(0.33)가 작아 보이는 것은 규제 때문이 아니라 단위가 커서 1 단위당 효과가 작기 때문이다(1 표준편차당으로 환산하면 약 3.9 로, 표준화 후 값 3.33 과 비슷하다). 오히려 스케일이 작은 noise1·noise2 가 같은 효과를 내려면 큰 계수가 필요해 페널티를 훨씬 강하게 받는다(noise2: 1 표준편차당 -1.15, 표준화 후 -6.07). 표준화 후에는 세 변수가 비슷한 크기(2~6 범위)로 정상화된다.

직관과 반대되는 결과라는 점이 중요합니다. “스케일이 큰 변수의 계수가 부풀려질 것”이라 예상하기 쉽지만, 실제로는 정반대입니다. 스케일이 큰 변수는 같은 예측력을 내는 데 작은 계수로 충분하므로, 페널티(계수 크기에 비례)의 영향을 상대적으로 덜 받습니다. 표준화를 생략하면 이런 왜곡이 모든 변수에 걸쳐 무작위로 발생해, 정작 중요한 변수의 계수가 억울하게 짓눌릴 수 있습니다.

표준화 코드

R: `scale(X)` 또는 `caret::preProcess(X, method=c("center","scale"))`

Python: `sklearn.preprocessing.StandardScaler()`

이 가이드의 기본 데이터셋(diabetes 원본 10 개 변수)은 이미 표준화되어 제공되지만, noise 변수를 추가했으므로 전체 데이터셋에 대해 다시 표준화를 적용합니다.

3-1. 페널티 항과 목적함수

Ridge 회귀는 계수의 제곱합에 페널티를 부과합니다(L2 규제라고도 부릅니다).

$$\text{목적함수} = \text{RSS} + \lambda \sum \beta_j^2$$

λ (람다)는 규제의 강도를 조절하는 하이퍼파라미터입니다. $\lambda=0$ 이면 일반 OLS 와 같아지고, λ 가 커질수록 계수들이 점점 0 에 가깝게 눌립니다(이를 축소, shrinkage 라 부릅니다). 다만 곧 보겠지만 Ridge 의 계수는 λ 가 아무리 커도 정확히 0 이 되지는 않습니다.

3-2. 기하학적 직관

Ridge 가 왜 계수를 0 에 가깝게는 만들어도 정확히 0 으로 만들지는 않는지, 그림으로 이해할 수 있습니다. 회귀계수가 2 개(β_1, β_2)뿐인 간단한 상황을 생각해 봅시다.

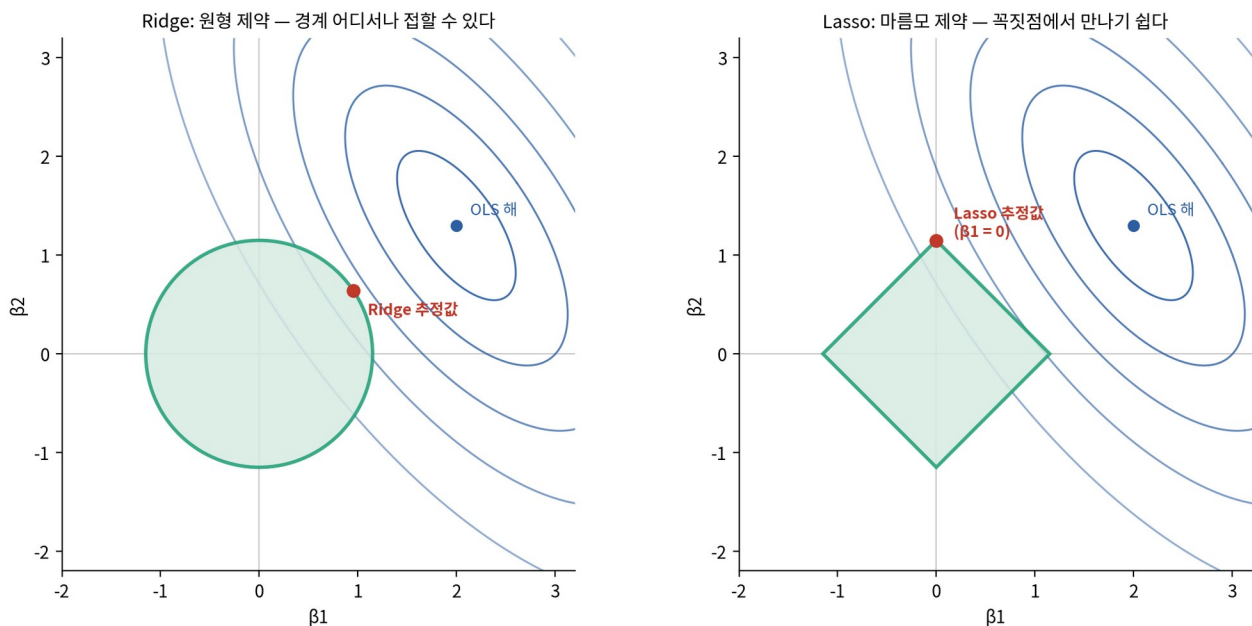


그림 3. 등고선(파란 타원)은 OLS 의 RSS 가 같은 값을 갖는 지점들을 이은 선으로, 중심(OLS 해)에서 멀어질수록 RSS 가 커진다. 규제는 계수가 초록색 제약 영역 안에 머물도록 강제한 뒤, 그 안에서 RSS 가 가장 작은 지점을 찾는 것과 같다. Ridge(왼쪽)의 제약 영역은 원이라 등고선과 어느 지점에서나 접할 수 있는 반면, Lasso(오른쪽)의 제약 영역은 마름모라 꼭짓점(축 위의 점)에서 접하기 쉽다.

Ridge 는 제약 영역이 원형입니다. 원에는 모서리가 없으므로, 타원형 등고선이 원과 접하는 지점은 일반적으로 축 위(즉 어느 계수가 정확히 0 인 지점)가 아니라 원의 둘레 어딘가입니다. 그 결과 Ridge 는 관련 있는 변수들의 계수를 고르게 줄이긴 하지만, 좀처럼 정확히 0 으로 만들지는 않습니다.

3-3. R / Python 코드

R

```
library(glmnet)

x <- scale(as.matrix(X)) # 반드시 표준화 후 행렬로 변환
y <- as.vector(y)

# alpha=0 이면 순수 Ridge (glmnet에서 alpha는 L1 비율을 의미, 5장 alpha와 다른 개념)
n <- nrow(x); sy <- sqrt(mean((y - mean(y))^2)) # y의 표준편차(n으로 나눔)
# glmnet은 내부에서 y를 표준화하므로, Python Ridge(alpha=10)과 같게 하려면 변환 필요
ridge_fit <- glmnet(x, y, alpha = 0, lambda = 10 * sy / n)
coef(ridge_fit)
```

Python

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train) # 훈련 데이터(80%)로만 학습

ridge = Ridge(alpha=10) # scikit-learn의 alpha =  $\lambda$ (규제 강도). R과 값을 맞추는 법은 아래
주의 박스 참고
ridge.fit(X_scaled, y_train)
print(ridge.coef_)
```

R과 Python의 용어 차이에 주의

R의 glmnet에서 “alpha”는 L1과 L2의 혼합 비율(이 가이드의 α , 5장에서 다룸)을 의미합니다.

Python의 scikit-learn에서 “alpha”는 규제 강도(이 가이드의 λ)를 의미합니다.

즉 같은 단어 “alpha”가 R과 Python에서 서로 다른 개념을 가리킵니다. 이 가이드에서는 혼동을 피하기 위해 규제 강도는 항상 λ , 혼합 비율은 항상 α 로 표기합니다.

또 하나 주의할 점이 있습니다. 같은 λ 값이라도 두 패키지의 목적함수 정의가 달라, 값을 그대로 옮기면 결과가 달라집니다. Lasso는 그대로 같지만, Ridge는 R에서 $\text{lambda} = (\text{Python의 alpha}) \times \text{sd}(y) / n$ 으로 바뀌어야 같은 모델이 됩니다(glmnet이 내부에서 y를 표준화하기 때문). 또 R의 scale()은 표본표준편차($n-1$)로 나누므로 소수점 아래에서 약간 차이가 날 수 있습니다(예: s1 계수 R -12.96, Python -13.23).

4. Lasso 회귀 (L1 규제)

4-1. 페널티 항과 목적함수

Lasso(Least Absolute Shrinkage and Selection Operator)는 계수의 절댓값 합에 페널티를 부과합니다(L1 규제).

$$\text{목적함수} = \text{RSS} + \lambda \sum |\beta_j|$$

형태는 Ridge 와 비슷해 보이지만, 제곱(?) 대신 절댓값(| |)을 쓴다는 작은 차이가 결과에서는 큰 차이를 만듭니다. Lasso 는 λ 가 충분히 크면 일부 계수를 정확히 0 으로 만듭니다 — 이는 사실상 그 변수를 모델에서 제거하는 것과 같습니다. 그래서 Lasso 는 규제인 동시에 자동으로 변수를 선택해 주는 도구로도 쓰입니다.

4-2. 기하학적 직관

3 장 그림 3 의 오른쪽 그림을 다시 보면, Lasso 의 제약 영역은 마름모(다이아몬드) 모양입니다. 마름모는 축 위에 뾰족한 꼭짓점을 가지고 있습니다. 타원형 등고선이 점점 커지다가 이 마름모와 처음 맞는 지점은, 모서리가 없는 원과 달리, 꼭짓점에서 만나는 경우가 비스듬한 변에서 만나는 경우보다 기하학적으로 훨씬 흔합니다. 꼭짓점은 정확히 한 축 위에 있는 점, 즉 한 계수는 0 이고 다른 계수만 값을 갖는 지점입니다.

변수가 여러 개로 늘어나면 제약 영역은 다이아몬드 모양의 고차원 입체(꼭짓점과 모서리가 더 많은 형태)가 되고, 등고선이 이 입체의 꼭짓점이나 모서리에서 만나는 경우가 흔해집니다. 이게 Lasso 가 일부 계수를 정확히 0 으로 보내는 기하학적 이유입니다.

4-3. R / Python 코드

R

```
library(glmnet)

x <- scale(as.matrix(X))
y <- as.vector(y)

# alpha=1 이면 순수 Lasso
lasso_fit <- glmnet(x, y, alpha = 1, lambda = 3)
coef(lasso_fit)
```

Python

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train) # 훈련 데이터(80%)로만 학습

lasso = Lasso(alpha=3, max_iter=20000) # max_iter를 충분히 크게
lasso.fit(X_scaled, y_train)
print(lasso.coef_)
```

4-4. Ridge와 Lasso의 차이 정리

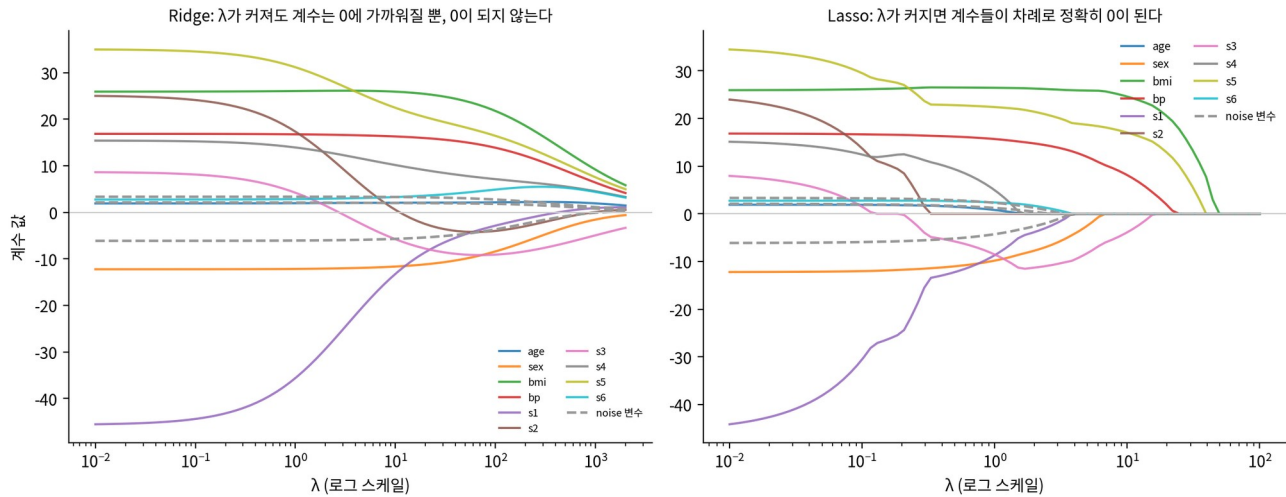


그림 4. λ 를 0에 가까운 값(왼쪽)에서 큰 값(오른쪽)으로 늘려갈 때 각 변수의 계수가 어떻게 변하는지 보여주는 “계수 경로”. Ridge(왼쪽 그래프)는 λ 가 커져도 모든 계수가 0을 향해 서서히 줄어든 뿐 정확히 0에 닿지 않는다. Lasso(오른쪽 그래프)는 λ 가 커짐에 따라 계수들이 한 번에 하나씩 정확히 0에 닿아 그래프에서 사라진다. 회색 점선은 noise 변수들로, Lasso에서는 비교적 이른 λ 구간에서 모두 0이 되는 반면 Ridge에서는 끝까지 작은 값으로 남는다.

	Ridge (L2)	Lasso (L1)
계수가 정확히 0이 되는가	되지 않는다	된다 (변수 선택 효과)
상관된 변수들을 다루는 방식	비슷한 변수끼리 계수를 나눠 가짐	비슷한 변수 중 하나만 임의로 선택
적합한 상황	변수 대부분이 어느 정도 중요할 때	일부 변수만 중요하고 나머지는 무관할 때
계산 안정성	해가 유일하게 결정됨(단한 해 존재)	상관된 변수가 많으면 선택이 불안정할 수 있음

마지막 행(“계산 안정성”)은 실무에서 자주 간과되는 부분입니다. s1과 s2처럼 상관관계가 강한 두 변수가 있을 때, Lasso는 둘 중 어느 쪽을 선택할지가 데이터의 미세한 차이(난수 시드, 표본 추출 방식)에 따라 달라질 수 있습니다. 이는 Lasso가 잘못 작동해서가 아니라, 거의 같은 정보를 담은 두 변수 중 “어느 쪽이든 상관없다”는 점에서는 합리적인 동작입니다. 다만 “이 변수가 빠졌으니 중요하지 않다”고 해석할 때는 주의가 필요합니다 — 빠진 변수가 다른 상관된 변수로 대체되었을 뿐일 수 있습니다. 이 문제를 누그러뜨리는 방법이 다음 장의 Elastic Net입니다.

5. Elastic Net (L1 + L2 혼합)

5-1. 왜 절충이 필요한가

4 장 끝에서 짚었듯, Lasso 는 상관관계가 높은 변수군을 만나면 그중 하나만 임의로 선택하고 나머지는 0 으로 보내는 경향이 있습니다. 두 변수가 실제로 둘 다 의미 있는 정보를 담고 있다면, 이렇게 한쪽을 통째로 버리는 것이 항상 바람직하지는 않습니다.

반대로 Ridge 는 상관된 변수들을 모두 살려두지만, 정작 명백히 무관한 변수의 계수도 정확히 0 으로 만들어주지는 못합니다.

Elastic Net 은 Ridge 의 페널티(L2)와 Lasso 의 페널티(L1)를 함께 사용해서, “상관된 변수를 그룹으로 유지하는 능력”과 “무관한 변수를 제거하는 능력”을 동시에 얻으려는 절충안입니다.

5-2. 페널티 항과 목적함수

Elastic Net 은 L1 과 L2 페널티를 α (알파)라는 비율로 섞습니다.

$$\text{목적함수} = \text{RSS} + \lambda [\alpha \sum |\beta_j| + (1-\alpha) \sum \beta_j^2]$$

α 는 0 과 1 사이의 값으로, L1 과 L2 를 얼마나 섞을지 정합니다.

- $\alpha = 1$: L1 페널티만 사용 — Lasso 와 완전히 같아집니다.
- $\alpha = 0$: L2 페널티만 사용 — Ridge 와 완전히 같아집니다.
- $0 < \alpha < 1$: 두 페널티가 섞인 절충 상태.

5-3. Ridge·Lasso 는 Elastic Net 의 특수한 경우

위 수식에서 바로 알 수 있듯, Ridge 와 Lasso 는 사실 Elastic Net 이라는 더 일반적인 틀 안에서 α 를 양 끝값(0 또는 1)으로 고정한 특수한 경우입니다. 처음부터 셋을 서로 다른 세 가지 방법으로 외우기보다, “Elastic Net 이라는 하나의 틀 안에서 L1 과 L2 의 비율(α)과 전체 강도(λ)라는 두 손잡이를 돌리는 것”으로 이해하면 세 방법의 관계가 한결 명확해집니다.

5-4. R / Python 코드

R

```
library(glmnet)

x <- scale(as.matrix(X))
y <- as.vector(y)

# alpha = L1 비율(이 가이드의  $\alpha$ ), lambda = 규제 강도(이 가이드의  $\lambda$ )
a_py <- 1.5; r <- 0.5 # Python ElasticNet(alpha=1.5, l1_ratio=0.5)
n <- nrow(x); sy <- sqrt(mean((y - mean(y))^2)) # y의 표준편차(n으로 나눔)
lam <- a_py * r + a_py * (1 - r) * sy # 약 59.2
en_fit <- glmnet(x, y, alpha = a_py * r / lam, lambda = lam) # alpha 약 0.0127
coef(en_fit)
```

Python

```

from sklearn.linear_model import ElasticNet
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train) # 훈련 데이터(80%)로만 학습

# alpha =  $\lambda$ (규제 강도), l1_ratio =  $\alpha$ (L1 비율) - R과 인자 이름이 다름
en = ElasticNet(alpha=1.5, l1_ratio=0.5, max_iter=20000)
en.fit(X_scaled, y_train)
print(en.coef_)

```

주의: R 과 Python 의 인자 이름이 또 한 번 다릅니다

R glmnet: $\alpha = \alpha$ (L1 비율), $\lambda = \lambda$ (규제 강도)

Python ElasticNet: $\alpha = \lambda$ (규제 강도), $\text{l1_ratio} = \alpha$ (L1 비율)

R 의 “alpha”와 Python 의 “alpha”가 서로 다른 것을 가리킨다는 점을 3 장에서 이미 짚었는데, Elastic Net 에서는 두 하이퍼파라미터가 동시에 등장하므로 더더욱 헷갈리기 쉽습니다. 코드를 옮겨 쓸 때 반드시 두 언어의 인자 이름을 따로 확인하십시오.

값도 변환해야 합니다. Python ElasticNet($\alpha=a$, $\text{l1_ratio}=r$)과 같은 모델을 R glmnet 으로 만들려면 $\lambda = a \cdot r + a \cdot (1-r) \cdot \text{sd}(y)$, $\alpha = a \cdot r / \lambda$ 로 넣어야 합니다. 이 가이드의 예제($a=1.5$, $r=0.5$)는 R 에서 $\alpha \approx 0.0127$, $\lambda \approx 59.2$ 가 됩니다. 반대로 R 코드에 $\alpha = 0.5$, $\lambda = 1.5$ 를 그대로 넣으면 전혀 다른 모델이 됩니다(s_1 계수 -9.82 vs -0.25).

6. 하이퍼파라미터 튜닝과 최종 비교

6-1. λ 는 무엇을 조절하는가

지금까지는 λ 값을 임의로(예: 10, 3, 1.5) 정해서 사용했습니다. 그런데 λ 를 너무 작게 잡으면 규제 효과가 거의 없어 OLS 와 다를 바 없이 과적합되고, 너무 크게 잡으면 계수가 지나치게 눌러 정작 중요한 패턴까지 놓치는 과소적합이 됩니다. 이는 2 장에서 본 편향-분산 트레이드오프와 그대로 연결됩니다.

- λ 가 작을수록: 모델이 데이터에 유연하게 맞춰짐 — 편향은 낮지만 분산이 높음 (과적합 위험)
- λ 가 클수록: 모델이 단순해짐 — 분산은 낮지만 편향이 높음 (과소적합 위험)

따라서 “적당한” λ 를 찾는 것이 실전에서 가장 중요한 작업입니다. 이를 사람이 감으로 정하지 않고 데이터로부터 체계적으로 찾는 방법이 교차검증(cross-validation, CV)입니다.

6-2. 교차검증(CV)으로 λ 찾기

교차검증의 기본 발상은 훈련 데이터를 여러 조각(통상 5 개 또는 10 개, fold)으로 나눈 뒤, 한 조각씩 돌아가며 검증용으로 떼어 놓고 나머지로 모델을 학습시켜 검증 성능을 측정하는 것입니다. 각 λ 후보마다 이 과정을 반복해 평균 성능을 비교하고, 평균 오차가 가장 낮은 λ 를 선택합니다.

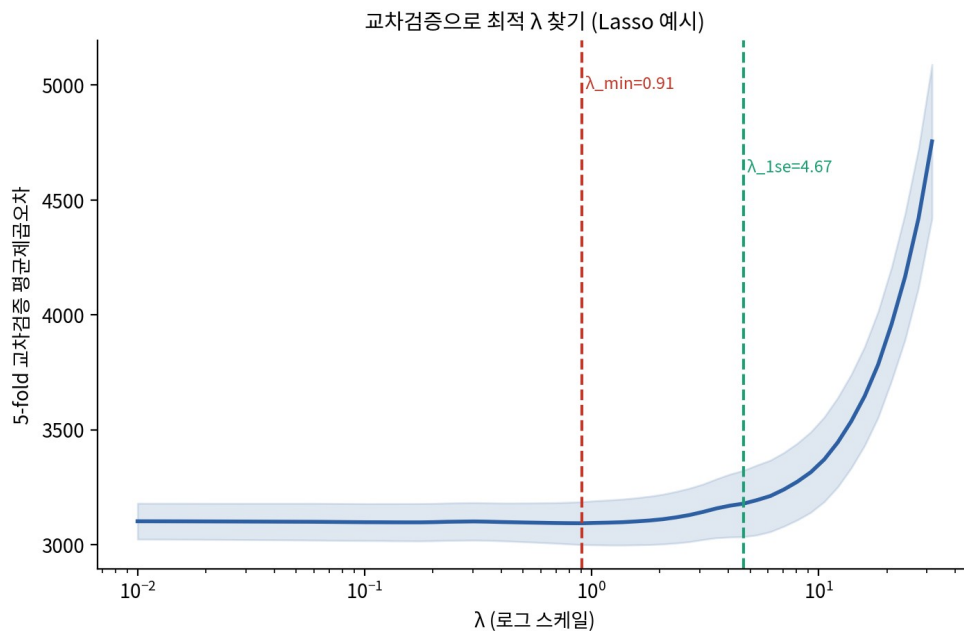


그림 5. Lasso 에 대해 5-fold 교차검증으로 λ 를 바꿔가며 측정한 평균제곱오차 곡선. 음영은 fold 간 표준오차 범위를 나타낸다. $\lambda_{\min}$ (빨강)은 평균 오차가 가장 낮은 지점이고, λ_{1se} (초록)은 $\lambda_{\min}$ 의 오차에서 표준오차 1 개만큼 여유를 두면서도 가장 단순한(λ 가 가장 큰) 모델을 선택하는 관행적 기준이다.

그림 5 에서 $\lambda_{\min}$ 과 λ_{1se} 라는 두 가지 후보가 보입니다. $\lambda_{\min}$ 은 검증 오차를 가장 작게 만드는 값이지만, 표본의 우연성에 따라 살짝 다르게 나올 수 있어 다소 불안정합니다. λ_{1se} 는 “ $\lambda_{\min}$ 과 비교해 오차가 표준오차 1 개 이내로만 나빠지는 한도 안에서, 가장 단순한(계수가 더 많이 0 이 되는, λ 가 더 큰) 모델”을 택하는 방법으로, 더 안정적이고 해석하기 쉬운 모델을 선호할 때 사용합니다.

R

```
library(glmnet)
```

```
x <- scale(as.matrix(X))
y <- as.vector(y)

set.seed(42)
cv_lasso <- cv.glmnet(x, y, alpha = 1, nfolds = 5)
# R은 fold 배정 방식이 Python과 달라  $\lambda_{\min}$ 이 정확히 같지는 않음

cv_lasso$lambda.min # 검증오차 최소 지점
cv_lasso$lambda.1se # 1-SE 규칙 지점
coef(cv_lasso, s = "lambda.1se")
```

Python

```
from sklearn.linear_model import LassoCV
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
import numpy as np

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train) # 훈련 데이터(80%)로만 학습

alphas = np.logspace(-2, 1.5, 60)
lasso_cv = LassoCV(alphas=alphas, max_iter=20000,
                   cv=KFold(n_splits=5, shuffle=True, random_state=42)) # 섞지
않으면 random_state가 무시됨
lasso_cv.fit(X_scaled, y_train)

print("최적 lambda:", lasso_cv.alpha_)
# scikit-learn은 1-SE 규칙을 기본 제공하지 않아 직접 계산해야 함
```

R과 Python의 차이: 1-SE 규칙

R의 `cv.glmnet()`은 `lambda.min`과 `lambda.1se`를 모두 기본으로 제공합니다.

Python의 `LassoCV`/`RidgeCV`/`ElasticNetCV`는 `lambda.min`에 해당하는 값만 자동으로 제공하며, 1-SE 규칙을 쓰려면 각 fold의 오차와 표준오차를 직접 계산해야 합니다.

6-3. Elastic Net의 α 까지 함께 탐색하기

Elastic Net은 λ 와 α 두 하이퍼파라미터를 동시에 가지므로, 둘을 함께 탐색해야 합니다. 이 지점에서 R과 Python의 구현 방식이 비대칭적이라는 점을 짚어야 합니다.

- R의 `cv.glmnet()`은 한 번 호출에 λ 만 자동으로 탐색합니다. α 까지 함께 찾으려면 몇 가지 α 후보를 직접 정해두고, 각 α 마다 `cv.glmnet()`을 반복 호출해 가장 좋은 조합을 손수 비교해야 합니다.
- Python의 `ElasticNetCV()`는 `l1_ratio` 인자에 후보 배열을 넘기면 λ 와 α (`l1_ratio`)를 한 번의 호출로 함께 탐색합니다.

이 차이 때문에 R 코드가 Python보다 한 단계 더 길어집니다. 둘 다 같은 일을 하지만 구현 방식이 다르다는 점을 미리 알아두면 코드를 옮길 때 헛갈리지 않습니다.

R

```
library(glmnet)
```

```

x <- scale(as.matrix(X)); y <- as.vector(y)
alpha_candidates <- c(0.1, 0.3, 0.5, 0.7, 0.9)

results <- lapply(alpha_candidates, function(a) {
  cv <- cv.glmnet(x, y, alpha = a, nfolds = 5)
  data.frame(alpha = a, lambda_min = cv$lambda.min,
             cvm = min(cv$cvm))
})
results_df <- do.call(rbind, results)
best <- results_df[which.min(results_df$cvm), ]
print(best) # 최적 alpha와 lambda 조합

```

Python

```

from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import KFold # X_scaled, y_train은 앞 코드에서 준비
import numpy as np

alphas = np.logspace(-2, 1.5, 60)
l1_ratios = [0.1, 0.3, 0.5, 0.7, 0.9]

en_cv = ElasticNetCV(l1_ratio=l1_ratios, alphas=alphas,
                    cv=KFold(n_splits=5, shuffle=True, random_state=42),
                    max_iter=20000)
en_cv.fit(X_scaled, y_train)

print("최적 lambda:", en_cv.alpha_)
print("최적 alpha(L1 비율):", en_cv.l1_ratio_)

```

6-4. 튜닝된 세 모델의 최종 비교

같은 데이터셋(훈련 80% / 테스트 20%로 분할, 표준화 적용)에 OLS, Ridge, Lasso, Elastic Net 을 각각 적합한 결과를 비교합니다. Ridge 는 $\lambda=10$, Lasso 는 $\lambda=3$, Elastic Net 은 $\lambda=1.5 \cdot \alpha=0.5$ 를 사용했습니다(세 방법의 차이를 뚜렷하게 보여주기 위해 고른 값으로, CV 최적값은 아닙니다. 예를 들어 6-2 절에서 구한 Lasso 의 $\lambda_{\min}$ 은 0.91 로, 여기서 쓴 3 보다 작습니다).

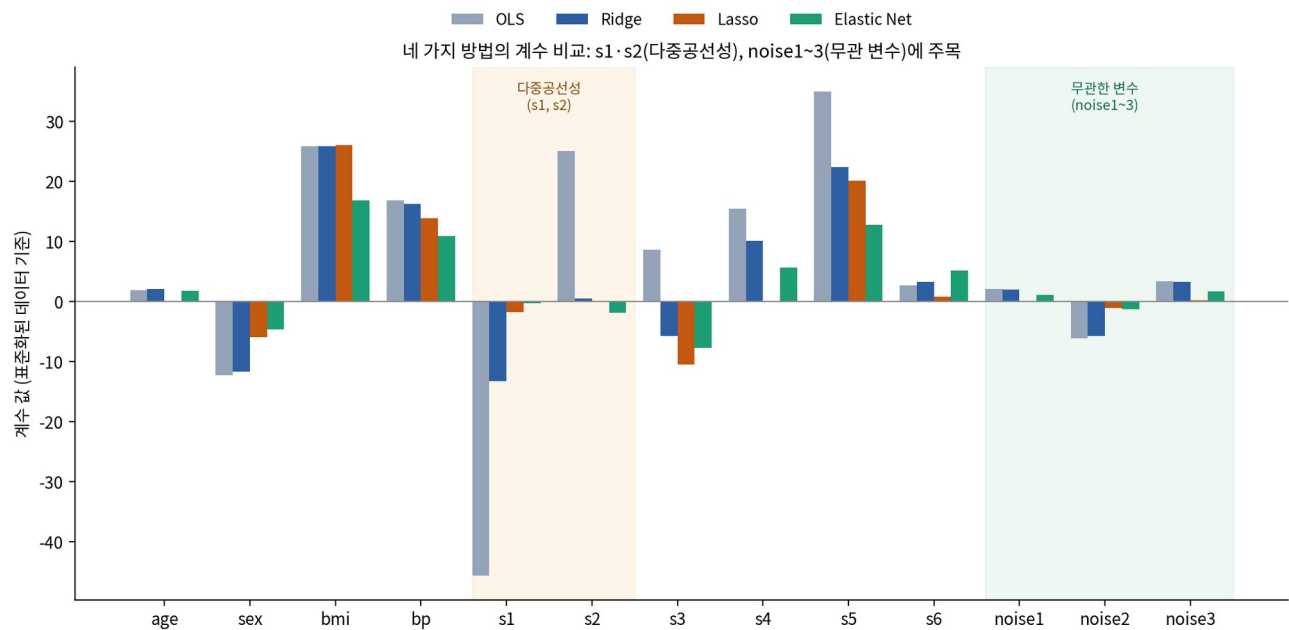


그림 6. 네 가지 방법의 계수 비교(표준화된 데이터 기준). 주황 영역(s1, s2)은 다중공선성이 있는 변수쌍이고, 초록 영역(noise1~3)은 target 과 무관한 변수다.

그림 6 에서 가장 눈에 띄는 부분은 s1 입니다. OLS 에서 s1 의 계수는 -45.66 으로 극단적으로 크고 음수인데, 이는 s1 이 실제로 중요해서가 아니라 s2 와 상관관계가 너무 강해서(0.90) 둘 사이에서 계수가 불안정하게 나뉘는 결과입니다(s2 는 반대로 +25.09 라는 큰 양수를 갖습니다). Ridge 는 이 둘을 각각 -13.23 과 +0.51 로 끌어내려 안정화시키고, Lasso 는 s2 를 사실상 0(-0.00)으로 보내고 s1 도 -1.83 으로 크게 줄입니다. Elastic Net 은 s1 과 s2 둘 다 작게 유지(-0.25, -1.92)하면서 완전히 0 으로 보내지는 않는, Ridge 와 Lasso 사이의 절충적 결과를 보여줍니다.

noise1~3 영역에서는 Lasso 가 noise1(0.00)과 noise2 는 작게 유지하지만 정확히는 0 이 되지 않은 변수도 있다는 점에 주의할 필요가 있습니다. 실제로 이번 설정에서 noise1 만 정확히 0 이 되었고, 나머지 noise2·noise3 는 작지만 0 이 아닌 값으로 남았습니다. 이는 “Lasso 가 무관한 변수를 항상 완벽하게 걸러낸다” 는 식의 과장된 기대를 경계해야 함을 보여줍니다. λ 를 더 키우면 더 많은 변수가 0 이 되지만, 그만큼 bmi 나 s5 처럼 실제로 중요한 변수의 계수도 함께 줄어드는 trade-off 가 있습니다.

방법	테스트셋 R ²	0 이 된 변수 수	핵심 특징
OLS	0.428	0	s1·s2 계수가 극단적으로 불안정
Ridge	0.438	0	모든 변수 유지, 계수를 고르게 축소
Lasso	0.470	4	변수 선택, 가장 높은 테스트 성능
Elastic Net	0.442	0	Ridge 와 Lasso 의 절충적 결과

이 표의 수치는 특정 λ 값을 고정해 비교한 결과이며, λ 를 어떻게 잡느냐에 따라 순위가 달라질 수 있다는 점은 감안해야 합니다. 다만 이 예제에서 Lasso 가 가장 높은 테스트 R² 를 보인 것은, 이 데이터셋에 실제로 무관한 변수(noise)가 섞여 있어 변수 선택이 도움이 되는 상황이었기 때문입니다. 모든 데이터셋에서 Lasso 가 항상 가장 좋은 성능을 낸다고 일반화할 수는 없습니다.

언제 어떤 방법을 쓸 것인가

- 변수 대부분이 어느 정도씩은 중요하다고 생각되고, 변수끼리 상관관계가 있다면 — Ridge
- 변수가 매우 많고 그중 일부만 실제로 중요할 것이라 예상되며, 해석 가능한 단순한 모델을 원한다면 — Lasso
- Lasso 를 쓰고 싶지만 상관된 변수들을 한쪽만 임의로 버리는 게 마음에 걸린다면 — Elastic Net
- 확신이 서지 않는다면 — 세 방법 모두 교차검증으로 비교해보고, 검증 성능이 가장 좋은 쪽을 선택하는 것이 가장 안전합니다.

이 가이드는 Ridge, Lasso, Elastic Net 의 원리와 R/Python 적용법, 교차검증을 통한 하이퍼파라미터 튜닝까지 다루었습니다. 로지스틱 회귀에 규제를 적용하는 방법, 그룹 단위로 변수를 선택하는 Group Lasso, 트리 기반 모델의 정규화 기법 등은 이 가이드의 범위를 벗어나며 후속 자료에서 다룰 수 있는 주제입니다.