Ridge Regression

Problem:

Generate a predictor vector **X** of length **n = 100** (random vector **X**), as well as a noise vector **ε** of length **n = 100**. Generate a response vector **Y** of length **n = 100** according to the following model:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3 + \epsilon_i$$

where:

- $\beta_0 = 50$
- $\beta_1 = 10$
- $\beta_2 = -20$
- $\beta_3 = 0.1$

Perform ridge regression using **X**, $\mathbf{X^2}$, $\mathbf{X^3}$, and $\mathbf{X^4}$ as predictors. Choose any two different values of **λ** (different from 0 and ∞). With each **λ**, perform ridge regression both **with** and **without** standardizing the predictors. Then, compare the results.

Note: No built-in functions are allowed.

In [1]:
```
import numpy as np
```

In [2]:
```
# set random seed for reproducibility
np.random.seed(403)
```

In [3]:
```
# define parameters
n = 100
x = np.random.rand(n)
epsilon = np.random.normal(0, 1, n)
b0 = 50
b1 = 10
b2 = -20
b3 = 0.1
```

In [4]:
```
# standardizing the data
x_mean = np.mean(x)
x_std = np.std(x)
x_standardized = (x - x_mean) / x_std
```

In [5]:
```
# generate Y
Y = b0 + b1*x + b2*x**2 + b3*x**3 + epsilon
```

In [6]:
```
def ridge_regression(X, Y, lmd):
    n, p = X.shape
    I = np.eye(p)
    beta_hat = np.linalg.inv(X.T @ X + lmd * I) @ X.T @ Y
    return beta_hat
```

In [7]:
```
# not standardizing the data
lmd0 = 0.1
b_ridge0 = ridge_regression(np.column_stack((np.ones(n), x, x**2, x**3)), Y, lmd0)
# predict using the estimated coefficients
Y_hat0 = np.column_stack((np.ones(n), x, x**2, x**3)) @ b_ridge0
# calculate the mse
mse0 = np.mean((Y - Y_hat0)**2)
print(f"mse with lambda = {lmd0} without standardizing the data: {mse0}")

# standardizing the data
b_ridge0_ = ridge_regression(np.column_stack((np.ones(n), x_standardized, x_standardiz
# predict using the estimated coefficients
Y_hat0_ = np.column_stack((np.ones(n), x_standardized, x_standardized**2, x_standardiz
# calculate the mse
mse0_ = np.mean((Y - Y_hat0_)**2)
print(f"mse with lambda = {lmd0} with standardizing the data: {mse0_}")
```

```
mse with lambda = 0.1 without standardizing the data: 1.0939062762083729
mse with lambda = 0.1 with standardizing the data: 1.0740449019290557
```

In [8]:
```
lmd1 = 0.01
# not standardizing the data
b_ridge1 = ridge_regression(np.column_stack((np.ones(n), x, x**2, x**3)), Y, lmd1)
# predict using the estimated coefficients
Y_hat1 = np.column_stack((np.ones(n), x, x**2, x**3)) @ b_ridge1
# calculate the mse
mse1 = np.mean((Y - Y_hat1)**2)
print(f"mse with lambda = {lmd1} without standardizing the data: {mse1}")

# standardizing the data
b_ridge1_ = ridge_regression(np.column_stack((np.ones(n), x_standardized, x_standardiz
# predict using the estimated coefficients
Y_hat1_ = np.column_stack((np.ones(n), x_standardized, x_standardized**2, x_standardiz
# calculate the mse
mse1_ = np.mean((Y - Y_hat1_)**2)
print(f"mse with lambda = {lmd1} with standardizing the data: {mse1_}")
```

```
mse with lambda = 0.01 without standardizing the data: 1.0742068107622686
mse with lambda = 0.01 with standardizing the data: 1.068984594996808
```

Conclusion

1. With both λ 0.1 and 0.01, the mse after standardizing X has dropped, indicating an improved perforamce of ridge regression with standarization.

2. By dropping λ from 0.1 to 0.01, the mse decreased, indicating a lower penalty on coefficients with less regularization, introducing smaller bias.

Lasso Regression

Problem:

Use the dataset you generated in Problem 1 and fit the model for the same set of predictors using **Lasso regression**.

Choose any two different values of **λ** (different from 0 and ∞).

With each **λ**, perform **Lasso regression** without standardizing the predictors. Then perform **Lasso regression** standardizing the predictors.

Questions:

- What can you conclude from these experiments?

Note: No built-in functions are allowed.

In [9]:
```
x.shape, Y.shape, x_standardized.shape
```

Out[9]:
```
((100,), (100,), (100,))
```

In [10]:
```
lmd0, lmd1
```

Out[10]:
```
(0.1, 0.01)
```

In [11]:
```
class lasso_reg:
    def __init__(self, lmd, tol=1e-4, max_iter=1000):
        self.lmd = lmd
        self.tol = tol
        self.max_iter = max_iter
        self.coef_ = None

    def fit(self, X, Y):
        n, p = X.shape
        b = np.zeros(p)
        b_old = np.zeros(p)

        X_t_X = X.T @ X

        for _ in range(self.max_iter):
            for j in range(p):
                # compute the partial residual
                residual = Y - X @ b + X[:, j] * b[j]

                # update coefficient using soft-thresholding
                rho = X[:, j].T @ residual
                b[j] = self._soft_threshold(rho / X_t_X[j, j], self.lmd)

            # early stop if converge
            if np.linalg.norm(b - b_old, ord=2) < self.tol:
                break

            b_old = b.copy()

        self.coef_ = b
```

```
    def _soft_threshold(self, rho, lmd):
        if rho > lmd:
            return rho - lmd
        elif rho < -lmd:
            return rho + lmd
        else:
            return 0

    def predict(self, X):
        return X @ self.coef_

    def cal_mse(self, X, Y):
        Y_hat = self.predict(X)
        return np.mean((Y - Y_hat)**2)
```

In [12]:

```
# case without standardizing the data
lasso0 = lasso_reg(lmd0)
lasso0.fit(np.column_stack((np.ones(n), x, x**2, x**3)), Y)
# calculate the mse
mse_lasso0 = lasso0.cal_mse(np.column_stack((np.ones(n), x, x**2, x**3)), Y)
print(f"mse with lambda = {lmd0} without standardizing the data: {mse_lasso0}")

lasso1 = lasso_reg(lmd1)
lasso1.fit(np.column_stack((np.ones(n), x, x**2, x**3)), Y)
# calculate the mse
mse_lasso1 = lasso1.cal_mse(np.column_stack((np.ones(n), x, x**2, x**3)), Y)
print(f"mse with lambda = {lmd1} without standardizing the data: {mse_lasso1}")
```

```
mse with lambda = 0.1 without standardizing the data: 1.1508305732256225
mse with lambda = 0.01 without standardizing the data: 1.0871402994665913
```

In [13]:

```
# case with standardizing the data
lasso0_ = lasso_reg(lmd0)
lasso0_.fit(np.column_stack((np.ones(n), x_standardized, x_standardized**2, x_standarc
# calculate the mse
mse_lasso0_ = lasso0_.cal_mse(np.column_stack((np.ones(n), x_standardized, x_standardi
print(f"mse with lambda = {lmd0} with standardizing the data: {mse_lasso0_}")

lasso1_ = lasso_reg(lmd1)
lasso1_.fit(np.column_stack((np.ones(n), x_standardized, x_standardized**2, x_standarc
# calculate the mse
mse_lasso1_ = lasso1_.cal_mse(np.column_stack((np.ones(n), x_standardized, x_standardi
print(f"mse with lambda = {lmd1} with standardizing the data: {mse_lasso1_}")
```

```
mse with lambda = 0.1 with standardizing the data: 1.1755404017481759
mse with lambda = 0.01 with standardizing the data: 1.070824122036093
```

In [14]:

```
# since unclear the effect of changing lambda, use train-test split to evaluate the mc
x_train = x[:80]
x_test = x[80:]
y_train = Y[:80]
y_test = Y[80:]
# standardizing the data
x_train_standardized = (x_train - np.mean(x_train)) / np.std(x_train)
x_test_standardized = (x_test - np.mean(x_train)) / np.std(x_train)
```

In [15]:

```
# case without standardizing the data
lasso0 = lasso_reg(lmd0)
lasso0.fit(np.column_stack((np.ones(80), x_train, x_train**2, x_train**3)), y_train)
```

```
# calculate train mse
mse_train_lasso0 = lasso0.cal_mse(np.column_stack((np.ones(80), x_train, x_train**2, x
# calculate test mse
mse_test_lasso0 = lasso0.cal_mse(np.column_stack((np.ones(20), x_test, x_test**2, x_te
print(f"train mse with lambda = {lmd0} without standardizing the data: {mse_train_lass
print(f"test mse with lambda = {lmd0} without standardizing the data: {mse_test_lasso0

lasso1 = lasso_reg(lmd1)
lasso1.fit(np.column_stack((np.ones(80), x_train, x_train**2, x_train**3)), y_train)
# calculate train mse
mse_train_lasso1 = lasso1.cal_mse(np.column_stack((np.ones(80), x_train, x_train**2, x
# calculate test mse
mse_test_lasso1 = lasso1.cal_mse(np.column_stack((np.ones(20), x_test, x_test**2, x_te
print(f"train mse with lambda = {lmd1} without standardizing the data: {mse_train_lass
print(f"test mse with lambda = {lmd1} without standardizing the data: {mse_test_lasso1
```

```
train mse with lambda = 0.1 without standardizing the data: 1.2612498747532779
test mse with lambda = 0.1 without standardizing the data: 0.6818476146548968
train mse with lambda = 0.01 without standardizing the data: 1.1930347305758442
test mse with lambda = 0.01 without standardizing the data: 0.6829818446647704
```

In [16]:

```
# case with standardizing the data
lasso0_ = lasso_reg(lmd0)
lasso0_.fit(np.column_stack((np.ones(80), x_train_standardized, x_train_standardized**
# calculate train mse
mse_train_lasso0_ = lasso0_.cal_mse(np.column_stack((np.ones(80), x_train_standardized
# calculate test mse
mse_test_lasso0_ = lasso0_.cal_mse(np.column_stack((np.ones(20), x_test_standardized,
print(f"train mse with lambda = {lmd0} with standardizing the data: {mse_train_lasso0_
print(f"test mse with lambda = {lmd0} with standardizing the data: {mse_test_lasso0_}'

lasso1_ = lasso_reg(lmd1)
lasso1_.fit(np.column_stack((np.ones(80), x_train_standardized, x_train_standardized**
# calculate train mse
mse_train_lasso1_ = lasso1_.cal_mse(np.column_stack((np.ones(80), x_train_standardized
# calculate test mse
mse_test_lasso1_ = lasso1_.cal_mse(np.column_stack((np.ones(20), x_test_standardized,
print(f"train mse with lambda = {lmd1} with standardizing the data: {mse_train_lasso1_
print(f"test mse with lambda = {lmd1} with standardizing the data: {mse_test_lasso1_}'
```

```
train mse with lambda = 0.1 with standardizing the data: 1.2858833371163871
test mse with lambda = 0.1 with standardizing the data: 0.6007472024541338
train mse with lambda = 0.01 with standardizing the data: 1.1760703588835029
test mse with lambda = 0.01 with standardizing the data: 0.6446689662376079
```

Conclusion

1. With both λ 0.1 and 0.01, either test and train set, the mse after standardizing X has dropped, indicating an improved perforamce of lasso regression with standarization.

2. By dropping λ from 0.1 to 0.01, the mse deviate differnetly in tain and test sets. MSE would decrease in train set and increase in test set when λ drops from 0.1 to 0.01. Indicating decrease λ would increase the overfitting problem.