Ridge Lasso 코드 확인

Import Data

import numpy as np import pandas as pd from sklearn.model_selection import train_test_split from sklearn.linear_model import LinearRegression, Ridge, Lasso import matplotlib.pyplot as plt from sklearn.metrics import mean_squared_error

In [5]:
```
data = pd.read_csv('https://github.com/YonseiESC/ESC-21SUMMER/blob/main/week3/HW_data/data.c
sv?raw=True')
```

In [7]:
```
data.head()
```

Out[7]:

	Age	Experience	Income	Family	CCAvg
0	25	1	49	4	1.6
1	45	19	34	3	1.5
2	39	15	11	1	1.0
3	35	9	100	1	2.7
4	35	8	45	4	1.0

In [8]:
```
data.isnull().sum() # 결측치 없음 확인
```

Out[8]:
```
Age           0
Experience    0
Income        0
Family        0
CCAvg         0
dtype: int64
```

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Age         2500 non-null   int64
 1   Experience  2500 non-null   int64
 2   Income      2500 non-null   int64
 3   Family      2500 non-null   int64
 4   CCAvg       2500 non-null   float64
dtypes: float64(1), int64(4)
memory usage: 97.8 KB
```

In [10]:
```
y = data['Income'] # 종속변수
X = data.drop(['Income'], axis = 1) # 독립변수
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, random_state = 1
000)
# train 70% test 30%, random_state 수행시마다 동일한 결과 얻기 위해 적용-여러번 수행하더라도 같은 레코
드를 추출함.
```

Linear Regression

In [14]:
```
reg = LinearRegression()
results1 = reg.fit(x_train, y_train) # X_train, Y_train으로 fit linear model
```

Out[14]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [13]:
```
reg.coef_
```

Out[13]:
```
array([-3.07793956,  2.89401562, -3.37220023, 16.09065086])
```

Ridge Regression

In [15]:
```
rreg = Ridge(alpha = 0) # alpha = Lambda
                        # Lambda=0이면 제약x LS-method와 같은 결과
rreg.fit(x_train, y_train)
```

Out[15]:
```
Ridge(alpha=0, copy_X=True, fit_intercept=True, max_iter=None, normalize=False,
      random_state=None, solver='auto', tol=0.001)
```

In [16]:
```
rreg.coef_
```

Out[16]:
```
array([-3.07793956,  2.89401562, -3.37220023, 16.09065086])
```

In [17]:
```
alpha = np.logspace(-3,3,7)
# np.logspace(start,stop,num=Number of samples to generate) : Return numbers spaced evenly o
n a log scale
# 10^-3부터 10^3까지 log space 간격으로 일정하게 7개 생성.
alpha
```

Out[17]:
```
array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])
```

In [20]:
```
df = []
acc_table = []

for i, a in enumerate(alpha):     # i;횟수, a;위에서 생성한 alpha 값
        rreg = Ridge(alpha=a).fit(x_train, y_train)
        df.append(pd.Series(np.hstack([rreg.intercept_, rreg.coef_])))
    # np.hstack ; Stack arrays in sequence horizontally (column wise).
    # intercept&coef : 0  // 1 2 3 4 -> ln[16]
        pred_y = rreg.predict(x_test)

df_ridge = pd.DataFrame(df,index = alpha).T
df_ridge

# 결과에서 lambda가 0.001에서 1000.000으로 커질수록
# 1 2 3 4에 해당하는 계수들의 값을 보면
# 점점 shrink된다. 0에 가까워짐. * 그림 참고 *
```

Out[20]:

	0.001	0.010	0.100	1.000	10.000	100.000	1000.000
0	132.296084	132.295649	132.291303	132.247877	131.817002	127.823048	105.704966
1	-3.077937	-3.077919	-3.077732	-3.075864	-3.057321	-2.884607	-1.883048
2	2.894014	2.893995	2.893806	2.891920	2.873198	2.698718	1.681685
3	-3.372199	-3.372192	-3.372122	-3.371422	-3.364435	-3.295822	-2.731156
4	16.090648	16.090622	16.090363	16.087768	16.061871	15.807207	13.634454

Lasso Regression

In [21]:
```
lreg = Lasso(alpha = 0 ) # alpha = Lambda # alpha = Lambda
                         # Lambda=0이면 제약x, LS-method와 같은 결과
lreg.fit(x_train, y_train)
```

```
C:\Users\User\anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: With alpha=0,
this algorithm does not converge well. You are advised to use the LinearRegression estimator

C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: Us
erWarning: Coordinate descent with no regularization may lead to unexpected results and is di
scouraged.
  positive)
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: Co
nvergenceWarning: Objective did not converge. You might want to increase the number of iterat
ions. Duality gap: 1105890.1320882086, tolerance: 373.84840920000005
  positive)
```

Out[21]:
```
Lasso(alpha=0, copy_X=True, fit_intercept=True, max_iter=1000, normalize=False,
      positive=False, precompute=False, random_state=None, selection='cyclic',
      tol=0.0001, warm_start=False)
```

In [22]:
```
lreg.coef_
```

Out[22]:
```
array([-3.07790231,  2.8939786 , -3.37220244, 16.09065156])
```

In [29]:
```
df = []
acc_table = []

for i, a in enumerate(alpha):   # i;횟수, a;위에서 생성한 alpha 값
        lreg = Lasso(alpha=a).fit(x_train, y_train)
        df.append(pd.Series(np.hstack([lreg.intercept_, lreg.coef_])))
        pred_y = lreg.predict(x_test)

df_lasso = pd.DataFrame(df,index = alpha).T
df_lasso
# 결과에서 lambda가 0.001에서 1000.000으로 커질수록
# 1 2 3 4에 해당하는 계수들의 값을 보면
# 점점 shrink된다. * 그림 참고 *
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: Co
nvergenceWarning: Objective did not converge. You might want to increase the number of iterat
ions. Duality gap: 3094.379392363131, tolerance: 373.84840920000005
  positive)
```

Out[29]:

	0.001	0.010	0.100	1.000	10.000	100.000	1000.000
0	132.261976	131.960877	128.945930	98.937749	54.569493	73.876	73.876
1	-3.076625	-3.065044	-2.949074	-1.794975	-0.134206	-0.000	-0.000
2	2.892703	2.881139	2.765340	1.612913	-0.000000	-0.000	-0.000
3	-3.371595	-3.366136	-3.311548	-2.765340	-0.000000	-0.000	-0.000
4	16.090400	16.088142	16.065558	15.839618	13.184919	0.000	0.000

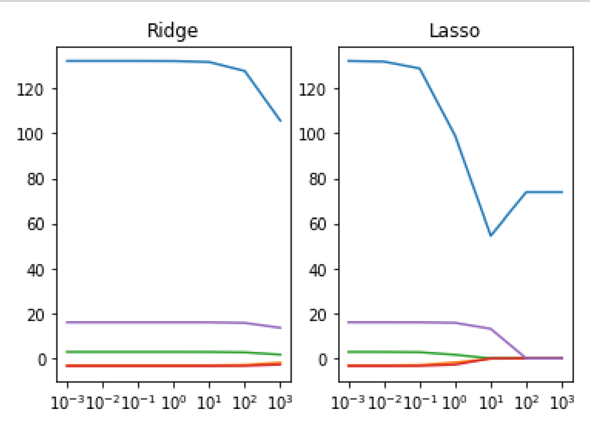
In [30]:
```
import matplotlib.pyplot as plt

ax1 = plt.subplot(121)
plt.semilogx(df_ridge.T)
plt.xticks(alpha)
plt.title("Ridge")

ax2 = plt.subplot(122)
plt.semilogx(df_lasso.T)
plt.xticks(alpha)
plt.title("Lasso")

plt.show()

# lambda가 클수록 0에 수렴하는 모습
# x축 lambda, y축 계수들의 값
```



ESL 3.29

!pip install IPython from IPython.display import Image

In [35]:
```
Image("C:/Users/User/Desktop/ESL3.29.jpg")
```

Out[35]:

Homework

Exercise. 3.29
Suppose we fit a ridge regression with a given shrinkage parameter $\lambda \in \mathbb{R}^+$ on a single variable x_1. (Notice that x_1 is a $N \times 1$ vector.)

1. (Essential) Show that the coefficient must be $\frac{X^T y}{X^T X + \lambda}$ where $X = x_1$.
2. (Essential) We now include an exact copy $x_2 = x_1$, so our new design matrix would be $X = [x_1|x_2]$. Using this matrix, re-fit our ridge regression. Show that both coefficients are identical, and derive their value.
3. (Extra) Show in general that if m copies of a variable x_j, are included in a ridge regression, so X would be $[x_1|x_2|\cdots|x_m]$, their coefficients are all the same.

sol1) Ridge $\min\left\{\sum_{i=1}(y_i-\hat{y}_i)^2+\lambda\sum_{j=1}^{p}\beta_j^2\right\}$

$\rightarrow \hat{\beta}^{ridge} = \mathrm{argmin}\left\{\|y-X\beta\|^2+\lambda\|\beta\|^2\right\}$

$\rightarrow \beta = \underbrace{(X^TX+\lambda I)^{-1}}_{\text{scalar }(\because x_1 \text{ is a } N\times 1 \text{ vector})} X^TY \Rightarrow \beta = \frac{X^Ty}{X^TX+\lambda}$

sol2) (p=2) $\|y-\beta X-\beta_2 X\|^2+\lambda\|\beta_1\|^2+\lambda\|\beta_2\|^2$ 최소

$\Rightarrow \sum_{i=1}^{n}(y_i-x_i\beta_1-x_i\beta_2)^2+\lambda\beta_1^2+\lambda\beta_2^2$

$\Rightarrow$ 편미분 $2\sum_{i=1}^{n}(y_i-x_i\beta_1-x_i\beta_2)(-x_i)+2\lambda\beta_1=0$

$2\sum_{i=1}^{n}(y_i-x_i\beta_1-x_i\beta_2)(-x_i)+2\lambda\beta_2=0$

$\Rightarrow X^T(y-X\beta_1-X\beta_2)=\lambda\beta_1$

$X^T(y-X\beta_1-X\beta_2)=\lambda\beta_2$

$\therefore \beta_1=\beta_2$

$\Rightarrow X^T(y-X\beta_1-X\beta_1)=\lambda\beta_1$

$X^T(y-2X\beta_1)=\lambda\beta_1$

$(\lambda+2X^TX)\beta_1=X^Ty$

$\therefore \beta_1=\frac{X^Ty}{\lambda+2X^TX}$

$\beta_1=\beta_2=\frac{X^Ty}{\lambda+2X^TX}$

---

sol3) (p=m) $\hat{\beta}^{ridge}=\mathrm{argmin}\left\{\|y-X\beta_1-X\beta_2-\cdots-X\beta_m\|^2+\lambda\|\beta_1\|^2+\lambda\|\beta_2\|^2+\cdots+\lambda\|\beta_m\|^2\right\}$

편미분 $\rightarrow$

$X^T(y-X\beta_1-X\beta_2-\cdots-X\beta_m)=\lambda\beta_1$

$X^T(y-X\beta_1-X\beta_2-\cdots-X\beta_m)=\lambda\beta_2$

$\vdots$

$X^T(y-X\beta_1-X\beta_2-\cdots-X\beta_m)=\lambda\beta_m$

$\Rightarrow X^T(y-mX\beta_1)=\lambda\beta_1$

$(\lambda+mX^TX)\beta_1=X^Ty$

$\therefore \beta_1=\frac{X^Ty}{\lambda+mX^TX}$

$\therefore \beta_1=\beta_2=\cdots=\beta_m$

$\beta_1=\beta_2=\cdots=\beta_m=\frac{X^Ty}{\lambda+mX^TX}$