Homework 2

1801212832 谌子民

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.1

In [2]:

```
def closed_form_1(x, y):
    x = np.column_stack((x, np.ones(x.shape[0])))
    x = np.mat(x)
    y = np.mat(y)
    theta = ((x.T * x).I) * x.T * y

    MSE = (y - x @ theta).T @ (y - x @ theta) / (x.shape[0] - x.shape[1])
    t_stat = theta.reshape(1, -1) / (np.sqrt(MSE[0][0] * np.diag(np.linalg.inv(x.T @
    return theta, t_stat
```

In [3]:

```
data = pd.read_csv('climate_change_1.csv')
data_train = data[data.Year <= 2006]
x_train = data_train.iloc[:, 2:-1].values
y_train = data_train.iloc[:, -1].values.reshape([len(data_train), 1])
data_test = data[data.Year > 2006]
x_test = data_test.iloc[:, 2:-1].values
y_test = data_test.iloc[:, -1].values.reshape([len(data_test), 1])
```

In [4]:

```
theta, t_stat = closed_form_1(x_train, y_train)
print('theta:')
print(theta)
print('t_stat:')
print(t_stat)
```

```
theta:
[[ 6.42053119e-02]
 [ 6.45735925e-03]
 [ 1.24041898e-04]
 [-1.65280032e-02]
 [-6.63048889e-03]
 [ 3.80810324e-03]
 [ 9.31410838e-02]
 [-1.53761324e+00]
 [-1.24594261e+02]]
t_stat:
[[ 9.92322578  2.8264197   0.2404694  -1.92972604 -4.07783387  3.75729
271
   6.31256095 -7.21030085 -6.26517396]]
```

1.2

Linear Model

$Y = X\beta + \varepsilon$

Definition of R square

$R^2 = \frac{SSR}{SST} = \frac{\sum(\breve{y}_i - \bar{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}$

In [5]:

```
def R_square(y_real, y_estimated):
    y_mean = np.mean(y_real)
    SST = np.sum((y_real - y_mean) ** 2)
    SSR = np.sum([x ** 2 for x in (y_mean - y_estimated)])
    return SSR / SST
```

In [6]:

```
# train set
y_estimated_train = np.column_stack((x_train, np.ones(x_train.shape[0]))) * theta
print(R_square(y_train, y_estimated_train))
```

```
0.7508932744079525
```

In [7]:

```
# test set
y_estimated_test = np.column_stack((x_test, np.ones(x_test.shape[0]))) * theta
print(R_square(y_test, y_estimated_test))
```

```
0.22517701397392234
```

1.3

In [8]:

```
t_stat
```

Out[8]:

```
matrix([[ 9.92322578,  2.8264197 ,  0.2404694 , -1.92972604, -4.077833
87,
          3.75729271,  6.31256095, -7.21030085, -6.26517396]])
```

From above result, we can see that MEI, CO2, CFC-11, CFC-12, TSI and Aerosols variables are significant.

1.4

Necessary conditions

$$(X'X)^{-1} exists$$

In [9]:

```
data_2 = pd.read_csv('climate_change_2.csv')
data_train_2 = data_2[data_2.Year <= 2006]
x_train_2 = data_train.iloc[:, 2:-1].values
y_train_2 = data_train.iloc[:, -1].values.reshape([len(data_train), 1])
data_test_2 = data_2[data_2.Year > 2006]
x_test_2 = data_test.iloc[:, 2:-1].values
y_test_2 = data_test.iloc[:, -1].values.reshape([len(data_test), 1])
```

In [10]:

```
theta, t_stat = closed_form_1(x_train_2, y_train_2)
print('theta:')
print(theta)
print('t_stat:')
print(t_stat)
```

```
theta:
[[ 6.42053119e-02]
 [ 6.45735925e-03]
 [ 1.24041898e-04]
 [-1.65280032e-02]
 [-6.63048889e-03]
 [ 3.80810324e-03]
 [ 9.31410838e-02]
 [-1.53761324e+00]
 [-1.24594261e+02]]
t_stat:
[[ 9.92322578  2.8264197   0.2404694  -1.92972604 -4.07783387  3.75729
271
   6.31256095 -7.21030085 -6.26517396]]
```

Why solution is unreasonable

$rank(X'X) = 9, whilesize(X) = 10 * 10$
$rank(X'X) < size(X)$

2.1

Loss function

$$L_1 : J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_0(x^{(i)} - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2]$$

$$L_2 : J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_0(x^{(i)} - y^{(i)})^2 + \lambda \sum_{j=1}^{n} |\theta_j|]$$

2.2

In [11]:

```
def closed_form_2(x, y, lamb):
    x = np.column_stack((x, np.ones(x.shape[0])))
    x = np.mat(x)
    y = np.mat(y)
    theta = ((x.T * x + lamb * np.eye(x.shape[1])).I) * x.T * y

    MSE = (y - x @ theta).T @ (y - x @ theta) / (x.shape[0] - x.shape[1])
    t_stat = theta.reshape(1, -1) / (np.sqrt(MSE[0][0] * np.diag(np.linalg.inv(x.T @
    return theta, t_stat
```

2.3

In [12]:

```
theta1, t_stat1 = closed_form_1(x_train, y_train)
print(theta1)
theta2, t_stat2 = closed_form_2(x_train, y_train, 10)
```

```
[[ 6.42053119e-02]
 [ 6.45735925e-03]
 [ 1.24041898e-04]
 [-1.65280032e-02]
 [-6.63048889e-03]
 [ 3.80810324e-03]
 [ 9.31410838e-02]
 [-1.53761324e+00]
 [-1.24594261e+02]]
```

In [13]:

```
print('The 2 norm of coefficients in OSL is: ', np.sum([x ** 2 for x in theta1[:-1]]
print('The 2 norm of coefficients in ridge regression is: ', np.sum([x ** 2 for x in
```

```
The 2 norm of coefficients in OSL is:  2.377425408787479
The 2 norm of coefficients in ridge regression is:  0.0026213992097659
875
```

The 2 norm of coefficients in ridge is much smaller, which means ridge regression is more robust.

2.4

In [14]:

```
lamb_set = [0.01, 0.1, 1, 10, 20]
R_square_train, R_square_test = [], []
for lamb in lamb_set:
    theta, t_stat = closed_form_2(x_train, y_train, lamb)
    y_estimated_train = x_train @ theta[:-1] + theta[-1]
    y_estimated_test = x_test @ theta[:-1] + theta[-1]
    R_square_train.append(R_square(y_train, y_estimated_train))
    R_square_test.append(R_square(y_test, y_estimated_test))

result = pd.DataFrame([R_square_train, R_square_test],
                      index = ['R square train', 'R square test'],
                     columns = lamb_set)
result
```

Out[14]:

	0.01	0.10	1.00	10.00	20.00
R square train	0.711653	0.694468	0.679469	0.674608	0.670752
R square test	0.585276	0.673288	0.846750	0.940872	0.985088

Use 5 fold cross validation method to decide the best lambda

In [15]:

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [16]:

```
model = LinearRegression()
kf = KFold(n_splits=5)
```

In [17]:

```
MSE_train, MSE_validation, MSE_test = [], [], []

for lamb in lamb_set:

    MSE_train_this = 0
    MSE_validation_this = 0
    MSE_test_this = 0

    for train_index, test_index in kf.split(y_train):

        x_train_sample, x_test_sample = x_train[train_index], x_train[test_index]
        y_train_sample, y_test_sample = y_train[train_index], y_train[test_index]

        theta, t_stat = closed_form_2(x_train_sample, y_train_sample, lamb)
        y_estimated_train = np.column_stack((x_train_sample, np.ones(x_train_sample.
        MSE_train_this += mean_squared_error(y_train_sample, y_estimated_train)

        y_estimated_test = np.column_stack((x_test_sample, np.ones(x_test_sample.sha
        MSE_validation_this += mean_squared_error(y_test_sample, y_estimated_test)

        MSE_test_this += mean_squared_error(y_test, np.column_stack((x_test, np.ones

    MSE_train.append(MSE_train_this / 5)
    MSE_validation.append(MSE_validation_this / 5)
    MSE_test.append(MSE_test_this / 5)
```

In [18]:

```
result = pd.DataFrame([MSE_train, MSE_validation, MSE_test],
                      index=['MSE_train', 'MSE_validation', 'MSE_test'],
                     columns=lamb_set)
```

In [19]:

```
result
```

Out[19]:

	0.01	0.10	1.00	10.00	20.00
MSE_train	0.008494	0.008690	0.009309	0.009548	0.009591
MSE_validation	0.034588	0.041579	0.052093	0.049333	0.045072
MSE_test	0.017495	0.019267	0.022420	0.024081	0.024853

As the result shows, 0.01 is the best value for lambda.

3.1 Workflow

1. calculate the correlation of different features
2. caculate the importance of features
3. drop the feature correlated features which are of less importance

3.2

In [20]:

```
result = pd.DataFrame(np.corrcoef(x_train.T),index=data.columns[2:-1],
                      columns=data.columns[2:-1])
result
```

Out[20]:

	MEI	CO2	CH4	N2O	CFC-11	CFC-12	TSI	Aerosols
MEI	1.000000	-0.041147	-0.033419	-0.050820	0.069000	0.008286	-0.154492	0.340238
CO2	-0.041147	1.000000	0.877280	0.976720	0.514060	0.852690	0.177429	-0.356155
CH4	-0.033419	0.877280	1.000000	0.899839	0.779904	0.963616	0.245528	-0.267809
N2O	-0.050820	0.976720	0.899839	1.000000	0.522477	0.867931	0.199757	-0.337055
CFC-11	0.069000	0.514060	0.779904	0.522477	1.000000	0.868985	0.272046	-0.043921
CFC-12	0.008286	0.852690	0.963616	0.867931	0.868985	1.000000	0.255303	-0.225131
TSI	-0.154492	0.177429	0.245528	0.199757	0.272046	0.255303	1.000000	0.052117
Aerosols	0.340238	-0.356155	-0.267809	-0.337055	-0.043921	-0.225131	0.052117	1.000000

There are high relationship among CO2, CH4, N2O, CFC-11 and CFC-12.

importance of different features

In [21]:

```
from sklearn.feature_selection import VarianceThreshold
select = VarianceThreshold(threshold=3).fit(x_train)
result = pd.DataFrame(select.variances_, index=data.columns[2:-1])
print(result)
```

```
                    0
MEI          0.861185
CO2        130.405741
CH4       2078.390657
N2O         22.563711
CFC-11     438.931351
CFC-12    3474.229454
TSI          0.160461
Aerosols     0.000898
```

N2O has has the biggest importance, so we drop CO2, CH4, CFC-11 and CFC-12.

In [22]:

```
new_features = ['MEI', 'N2O', 'TSI', 'Aerosols']
x_train_new = data_train.loc[:, new_features].values
y_train_new = data_train.iloc[:, -1].values.reshape([len(data_train), 1])
theta, t_stat = closed_form_1(x_train_new, y_train_new)

# train set
y_estimated_train_new = np.column_stack((x_train_new, np.ones(x_train_new.shape[0])))
print(R_square(y_train_new, y_estimated_train_new))
```

```
0.7261321279922042
```

4 Gradient Descent

Iterative expression

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

Loss function (Linear model)

In [23]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative
```

In [24]:

```
def loss_func(x):
    return (x-2.5)**2-1

def dLF(theta):
    return dericative(loss_func, theta, dx=1e-6)

def gradient_descent(theta0, alpha, epsilon=1e-6):
    theta = theta0
    while True:
        gradient = dLF(theta)
        last_theta = theta
        theta = theta - alpha * gradient
        if(abs(loss_func(theta) - loss_func(last_theta)) < epsilon):
            break
    return theta
```