In [1]:
```
#encoding:utf-8
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as pyplot
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
import sklearn


#[Problem 1.1 & 1.2]

def closed_form_1():
    #Create a general linear fit with intercept terms
    dataset = pd.read_csv("./data/climate_change_1.csv")
    X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI
","Aerosols"])

    X = np.column_stack((X,np.ones(len(X))))
    z = np.linalg.matrix_rank(X)
    print("The rank of the X matrix of climate1 is: ",z)
    print("The conditions for the X matrix of climate1 are: ",np.lin
alg.cond(X))
    y = dataset.get("Temp")
    if z != X.shape[1]:
        print("climate_1 matrix is unfilled rank, so it cannot be u
sed for linear models")
    X_train = X[:284]
    X_test = X[284:]
    y_train = y[:284]
    y_test = y[284:]

    X_train=np.mat(X_train)
    y_train = np.mat(y_train).T
    xTx = X_train.T*X_train
    w = 0
    if np.linalg.det(xTx)==0.0:
        print("xTx irreversible")
    else:

        w = np.ravel(xTx.I*(X_train.T*y_train))

    coef_=w[:-1]
    intercept_=w[-1]

    X_train=X_train[:,0:8]
    X_test = X_test[:,0:8]
    d = 0
    for i in range(8):
        d += coef_[i]*X_train[:,i]
```

```
    y_train_pred = d+intercept_

    s = 0
    for i in range(8):
        s += coef_[i]*X_test[:,i]
    y_test_pred = s+intercept_


    X_train = np.ravel(X_train).reshape(-1,8)
    y_train = np.ravel(y_train)
    y_train_pred = np.ravel(y_train_pred)

    print("Coefficient: ",coef_)
    print("Intercept: ",intercept_)
    print("the model is:  y = ",coef_,"* X +(",intercept_,")")

    y_train_avg = np.average(y_train)
    R2_train = np.sum((y_train_pred-y_train_avg)**2)/(np.sum((y_tra
in-y_train_avg)**2))
    print("R2 in Train :  ",R2_train)

    y_test_avg = np.average(y_test)
    R2_test = np.sum((y_test_pred-y_test_avg)**2)/(np.sum((y_test-y
_test_avg)**2))
    print("R2 in Test :  ",R2_test)

    dataset = pd.read_csv("./data/climate_change_2.csv")
    X_2 = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","T
SI","Aerosols","NO"])
    X_2 = np.column_stack((X,np.ones(len(X_2))))

    z = np.linalg.matrix_rank(X_2)
    print("The rank of the X matrix of climate2 is: ",z)
    print("The conditions for the X matrix of climate2 are: ",np.lin
alg.cond(X_2))
    if z != X_2.shape[1]:
        print("climate_2 matrix is unfilled rank, so it cannot be u
sed for linear models")

#[Problem 1.3]:According to the coefficient result, the most signif
icant is Aerosols, others is TSI>MEI>N2O
#[Problem 1.4]The applicable conditions of ordinary linear regressi
on
#1. The number of samples should be greater than the number of feat
ures
#2. The determinant of XT*X is not equal to 0
#3. X is a non-singular matrix

#[Problem2.1]
#Loss function for linear model with L1 regularization:JR(w)=0.5*|y
-Xw|**2 +λ∑|wi|
#Loss function for linear model with L2 regularization:JR(w)=0.5*|y
-Xw|**2 +0.5* λ|w|**2
```

```python
#[Problem2.2&2.4]
def closed_form_2():
    #Create L2 regular linear fitting with intercept term, i.e., ridge regression
    dataset = pd.read_csv("./data/climate_change_1.csv")
    X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols"])

    y = dataset.get("Temp")

    X = np.column_stack((X,np.ones(len(X))))

    for lamida in [10,1,0.1,0.01,0.001]:
        X_train = X[:284]
        X_test = X[284:]
        y_train = y[:284]
        y_test = y[284:]

        X_train=np.mat(X_train)
        y_train = np.mat(y_train).T
        xTx = X_train.T*X_train
        w = 0
        print("="*25+"L2 Redulatization (lamida is "+str(lamida)+")"+"="*25)
        I_m= np.eye(X_train.shape[1])
        if np.linalg.det(xTx+lamida*I_m)==0.0:
            print("xTx irreversible")
        else:

            w= (xTx+lamida*I_m).I*(X_train.T*y_train)

        wights = np.ravel(w)
        y_train_pred = np.ravel(np.mat(X_train)*np.mat(w))
        y_test_pred = np.ravel(np.mat(X_test)*np.mat(w))
        coef_=wights[:-1]
        intercept_=wights[-1]

        X_train=X_train[:,0:8]
        X_test = X_test[:,0:8]
        d = 0
        for i in range(8):
            d += coef_[i]*X_train[:,i]
        y_train_pred = d+intercept_

        s = 0
        for i in range(8):
            s += coef_[i]*X_test[:,i]
        y_test_pred = s+intercept_


        X_train = np.ravel(X_train).reshape(-1,8)
        y_train = np.ravel(y_train)
        y_train_pred = np.ravel(y_train_pred)
```

```
        y_train_avg = np.average(y_train)
        R2_train = np.sum((y_train_pred-y_train_avg)**2)/(np.sum((y
_train-y_train_avg)**2))
        print("R2 in Train :  ",R2_train)

        y_test_avg = np.average(y_test)
        R2_test = np.sum((y_test_pred-y_test_avg)**2)/(np.sum((y_te
st-y_test_avg)**2))
        print("R2 in Test :  ",R2_test)

        print("Coefficient: ",coef_)
        print("Intercept: ",intercept_)
        #The following is the formula of linear fitting:
        print("the model is:  y = ",coef_,"* X +(",intercept_,")")

#Cross-validation selects parameters
def build_model(x,y):
    kfold = KFold(n_splits=5).split(x, y)
    model = Ridge(normalize=True)
    lamibda_range = [10,1,0.1,0.01,0.001]
    grid_param = {"alpha":lamibda_range}

    grid = GridSearchCV(estimator=model,param_grid=grid_param,cv=kf
old,scoring="r2")
    grid.fit(x,y)
    print(grid.best_params_)
    return grid.best_params_


closed_form_1()
closed_form_2()

dataset = pd.read_csv("./data/climate_change_1.csv")
X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","A
erosols"])
y = dataset.get("Temp")

build_model(X,y)


# [Problem2.3]
# Regularization eliminates collinearity between features by increa
sing penalty functions. It can be understood as adding an L2 regula
r term to the linear regression loss function to limit the theta. B
y determining the value of lamida
# can balance the model between bias and variance. Adding lamibda*
identity matrix to the xTx matrix can make the determinant of the m
atrix whose determinant is close to 0 not equal to 0.
# is judged by r2, and fits best when lambda is 10.
# But it doesn't make sense to simply judge this, so take the cross
validation approach, such as build_model
# Because in the actual training, the fitting degree of the trainin
```

```
g results to the training set is usually good (the initial conditio
n is sensitive), but the fitting degree to the data outside the tra
ining set is usually not so satisfactory. So we usually don't
# will not use all data sets for training, but will separate a part
(this part does not participate in training) to test the parameters
generated by the training set, and relatively objectively judge the
consistency of these parameters to the data outside the training se
t.
# This idea is called Cross Validation.
# As the title suggests, specifying the training set and test machi
ne can skew the resulting model.
```

```
The rank of the X matrix of climate1 is:  9
The conditions for the X matrix of climate1 are:  8579851.99905957
7
Coefficient:  [ 6.42053134e-02  6.45735927e-03  1.24041895e-04 -1.
65280032e-02
 -6.63048889e-03  3.80810324e-03  9.31410838e-02 -1.53761324e+00]
Intercept:  -124.5942608183571
the model is:  y =  [ 6.42053134e-02  6.45735927e-03  1.24041895e-
04 -1.65280032e-02
 -6.63048889e-03  3.80810324e-03  9.31410838e-02 -1.53761324e+00]
* X +( -124.5942608183571 )
R2 in Train :   0.750893277277761
R2 in Test :   0.22517701758658804
The rank of the X matrix of climate2 is:  9
The conditions for the X matrix of climate2 are:  5.06656440892664
85e+22
climate_2 matrix is unfilled rank, so it cannot be used for linear
models
=========================L2 Redulatization (lamida is 10)=========
================
R2 in Train :   0.6746079231198266
R2 in Test :   0.9408716921404042
Coefficient:  [ 0.04054315  0.00814554  0.00020508 -0.01608137 -0.
00636145  0.003689
  0.00126458 -0.02443305]
Intercept:  -0.00022022058288633274
the model is:  y =  [ 0.04054315  0.00814554  0.00020508 -0.016081
37 -0.00636145  0.003689
  0.00126458 -0.02443305] * X +( -0.00022022058288633274 )
=========================L2 Redulatization (lamida is 1)==========
===============
R2 in Train :   0.6794692110083876
R2 in Test :   0.8467501181258112
Coefficient:  [ 0.04395558  0.00804313  0.00021395 -0.01693027 -0.
00646627  0.00376881
  0.00146759 -0.21177258]
Intercept:  -0.0022945422838525635
the model is:  y =  [ 0.04395558  0.00804313  0.00021395 -0.016930
27 -0.00646627  0.00376881
  0.00146759 -0.21177258] * X +( -0.0022945422838525635 )
=========================L2 Redulatization (lamida is 0.1)========
=================
```

```
R2 in Train :   0.6944684109168708
R2 in Test :   0.6732879127474877
Coefficient:  [ 5.06851277e-02  6.98925378e-03  1.30761990e-04 -1.
48156599e-02
 -6.07864608e-03  3.66100278e-03  1.36118274e-03 -8.71332452e-01]
Intercept:  -0.025045661913281534
the model is: y =  [ 5.06851277e-02  6.98925378e-03  1.30761990e-
04 -1.48156599e-02
 -6.07864608e-03  3.66100278e-03  1.36118274e-03 -8.71332452e-01]
* X +( -0.025045661913281534 )
=========================L2 Redulatization (lamida is 0.01)=======
==================
R2 in Train :   0.711652961739347
R2 in Test :   0.5852763146468936
Coefficient:  [ 5.46344723e-02  6.35012916e-03  7.94610956e-05 -1.
34794077e-02
 -5.83699154e-03  3.59093203e-03  1.44947810e-03 -1.26505174e+00]
Intercept:  -0.26232414556713424
the model is: y =  [ 5.46344723e-02  6.35012916e-03  7.94610956e-
05 -1.34794077e-02
 -5.83699154e-03  3.59093203e-03  1.44947810e-03 -1.26505174e+00]
* X +( -0.26232414556713424 )
=========================L2 Redulatization (lamida is 0.001)======
===================
R2 in Train :   0.714833043329129
R2 in Test :   0.5625217961259866
Coefficient:  [ 5.53981612e-02  6.25686043e-03  7.26293229e-05 -1.
33359358e-02
 -5.81554289e-03  3.58444627e-03  3.15485922e-03 -1.32868779e+00]
Intercept:  -2.592469636355677
the model is: y =  [ 5.53981612e-02  6.25686043e-03  7.26293229e-
05 -1.33359358e-02
 -5.81554289e-03  3.58444627e-03  3.15485922e-03 -1.32868779e+00]
* X +( -2.592469636355677 )
{'alpha': 0.1}
```

Out[1]:
```
{'alpha': 0.1}
```

In [2]:
```python
#encoding:utf-8
import numpy as np
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation
_factor
from sklearn import linear_model

# [Problem 3.1] remove features with high linearity according to VI
F
# VIF variance inflation factor, which measures the linearity betwe
en features
def vif(X, thres=1000):#The threshold is set to 1000
    col = list(range(X.shape[1]))
    dropped = True#If drop == True, then all calculated vifs are wi
thin the threshold and the traversal stops
```

```
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:,col].values, ix)
for ix in range(X.iloc[:,col].shape[1])]#Calculate the VIF values f
or eight variables
        maxvif = max(vif)
        maxix = vif.index(maxvif)
        if maxvif > thres:
            print('delete=',X.columns[col[maxix]],'  ', 'vif=',maxv
if )
            del col[maxix]
            dropped = True
    print('Remain Variables:', list(X.columns[col]))
    print('VIF:', vif)
    return list(X.columns[col])

'''
[Problem 3.2]The following program is a model for simple linear reg
ression based on the above results
'''
def closed_form_1():
    #Create a general linear fit with intercept terms
    dataset = pd.read_csv("./data/climate_change_1.csv")
    X = dataset.get(['MEI', 'CO2', 'CFC-11', 'CFC-12', 'Aerosols'])
#The result of the above selection

    X = np.column_stack((X,np.ones(len(X))))

    y = dataset.get("Temp")

    X_train = X[:284]
    X_test = X[284:]
    y_train = y[:284]
    y_test = y[284:]

    X_train=np.mat(X_train)
    y_train = np.mat(y_train).T
    xTx = X_train.T*X_train
    w = 0
    if np.linalg.det(xTx)==0.0:
        print("xTx irreversible")
    else:
        w = np.ravel(xTx.I*(X_train.T*y_train))

    coef_=w[:-1]
    intercept_=w[-1]

    X_train=X_train[:,0:5]
    X_test = X_test[:,0:5]
    d = 0
    for i in range(5):
        d += coef_[i]*X_train[:,i]
    y_train_pred = d+intercept_
```

```
    s = 0
    for i in range(5):
        s += coef_[i]*X_test[:,i]
    y_test_pred = s+intercept_


    X_train = np.ravel(X_train).reshape(-1,5)
    y_train = np.ravel(y_train)
    y_train_pred = np.ravel(y_train_pred)

    print("Coefficient: ",coef_)
    print("Intercept: ",intercept_)
    print("the model is:  y = ",coef_,"* X +(",intercept_,")")

    y_train_avg = np.average(y_train)

    R2_train = np.sum((y_train_pred-y_train_avg)**2)/(np.sum((y_tra
in-y_train_avg)**2))
    print("R2 in Train :  ",R2_train)

    y_test_avg = np.average(y_test)
    R2_test = np.sum((y_test_pred-y_test_avg)**2)/(np.sum((y_test-y
_test_avg)**2))
    print("R2 in Test :  ",R2_test)




closed_form_1()
```

```
Coefficient:  [ 0.05546626  0.00468232 -0.00374285  0.00232944 -1.
35511996]
Intercept:  -1.6456359522949424
the model is:  y =  [ 0.05546626  0.00468232 -0.00374285  0.002329
44 -1.35511996] * X +( -1.6456359522949424 )
R2 in Train :   0.7139899733846997
R2 in Test :   0.8240559664289709
```

In [ ]:

```
#encoding:utf-8
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# The goal of optimization is to minimize the loss function. The gr
adient direction of the function represents the direction where the
value of the function grows fastest.
# So the opposite direction is the direction in which the function
decreases the fastest. The optimal idea for gradient descent is to
use the negative gradient of the current position
# Direction as the search direction, also known as the "steepest de
scent method." Gradient descent method is an iterative algorithm, e
ach step needs to solve the objective function and gradient vector.
```

```
# steps:
# 1. Find the gradient,
# 2. Move in the direction opposite to the gradient, as follows, wh
ere, is the step size. If the step size is small enough, it can be
guaranteed that every iteration is decreasing, but it may cause the
convergence to be too slow. If the step size is too large, it canno
t guarantee that every iteration is decreasing, nor can it guarante
e the convergence.
# 3. Loop iteration step 2, until the value changes so that the dif
ference between the two iterations is small enough, such as 0.00000
001. In other words, until the value calculated by the two iteratio
ns is basically unchanged, then the local minimum value has been re
ached.
# 4. In this case, the output is the value that causes the function
to be minimal
# implementation process:
# loss function

def costFunc(X,Y,theta):
    #cost func
    inner=np.power((X*theta.T)-Y,2)
    return np.sum(inner)/(2*len(X))

def gradientDescent(X,Y,theta,alpha,iters):
    temp = np.mat(np.zeros(theta.shape))
    cost = np.zeros(iters)
    thetaNums = int(theta.shape[1])

    for i in range(iters):
        error = (X*theta.T-Y)
        for j in range(thetaNums):
            derivativeInner = np.multiply(error,X[:,j])
            temp[0,j] = theta[0,j]-(alpha*np.sum(derivativeInner)/l
en(X))#Compute the theta matrix
        theta = temp
        cost[i]=costFunc(X,Y,theta)
    return theta,cost


dataset = pd.read_csv("./data/climate_change_1.csv")
X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","A
erosols"])

y = dataset.get("Temp")
X = np.column_stack((np.ones(len(X)),X))
X_train = X[:284]
X_test = X[284:]
y_train = y[:284]
y_test = y[284:]

X_train = np.mat(X_train)
Y_train = np.mat(y_train).T

for i in range(1,9):
```

```
    X_train[:,i] = (X_train[:,i] - min(X_train[:,i])) / (max(X_trai
n[:,i]) - min(X_train[:,i]))

theta_n = (X_train.T*X_train).I*X_train.T*Y_train
print("theta =",theta_n)
theta = np.mat([0,0,0,0,0,0,0,0,0])
iters = 100000 # The number of iterations
alpha = 0.001 # learning rate

finalTheta,cost = gradientDescent(X_train,Y_train,theta,alpha,iters
)
print("final theta ",finalTheta)
print("cost ",cost)

fig, bx = plt.subplots(figsize=(8,6))
bx.plot(np.arange(iters), cost, 'r')
bx.set_xlabel('Iterations')
bx.set_ylabel('Cost')
bx.set_title('Error vs. Training Epoch')
plt.show()
# As the number of iterations increases, the loss function becomes
smaller and smaller, the trend becomes more and more stable, and th
e optimal solution is approached at the same time
```

```
theta = [[-0.07698894]
 [ 0.29450977]
 [ 0.28935427]
 [ 0.02211171]
 [-0.27724073]
 [-0.53156629]
 [ 0.7376296 ]
 [ 0.17604596]
 [-0.22725924]]
```