Homework 3

Deming Liu 1801212889

Problem 1:

(1)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns

print "Problem 1"
df = pd.read_csv('climate_change_1.csv')
df['const']=1
cols = list(df)
cols.insert(10, cols.pop(cols.index('const')))
df = df.loc[:,cols]
cols = list(df)
Training_set = df.loc[df['Year'] <= 2006]
Testing_set = df.loc[df['Year'] > 2006]
X = np.matrix(Training_set.iloc[:, 2:11].values)
Y = np.matrix(Training_set.iloc[:, 11].values).T
X_test = np.matrix(Testing_set.iloc[:, 2:11].values)
Y_test = np.matrix(Testing_set.iloc[:, 11].values)

def closed_form_1(X, Y):
    Theta = np.dot(np.dot(np.dot(X.T, X).I, X.T), Y)
    return Theta
```

(2)

```python
Theta = closed_form_1(X, Y)
temp_est = np.dot(X, Theta)
temp_est_test = np.dot(X_test, Theta)

pt_str = "Y = "
for idx in range(2, 10):
    pt_str += str(Theta[idx-2, 0])+"*"+str(cols[idx])+" + "
pt_str += str(Theta[8, 0])
print pt_str

def R_square(temp_est, Y):
    var_xb = (temp_est - temp_est.mean()).var()
    var_y = (Y - temp_est.mean()).var()
    R2 = var_xb / var_y
    return R2

R2 = R_square(temp_est,Y)
R2_test = R_square(temp_est_test, Y_test)
print "R2_training_set:", R2
print "R2_testing_set:", R2_test
```

```
Problem 1
Y = 0.0642053134626426*MEI + 0.006457359277697772*CO2 + 0.00012404189628771467*CH4 + -0.016528003337858137*N2O + -0.006630488906684447*CFC-11 + 0.003808103250738935*CFC-12 + 0.09314108447268625*TSI +
-1.5376132402599452*Aerosols + -124.59426172779982
R2_training_set: 0.7508932778196102
R2_testing_set: 0.18739109685216868
[Finished in 1.3s]
```

(3)

```python
def R_square(temp_est, Y):
    var_xb = (temp_est - temp_est.mean()).var()
    var_y = (Y - temp_est.mean()).var()
    R2 = var_xb / var_y
    return R2

R2 = R_square(temp_est,Y)
R2_test = R_square(temp_est_test, Y_test)
print "R2_training_set:", R2
print "R2_testing_set:", R2_test

reg = sm.OLS(endog=Training_set['Temp'], exog=Training_set[['MEI', 'CO2','CH4','N2O','CFC-11','CFC-12','TSI','Aerosols','const']], missing='drop')
results = reg.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Temp   R-squared:                       0.751
Model:                            OLS   Adj. R-squared:                  0.744
Method:                 Least Squares   F-statistic:                     103.6
Date:                Fri, 27 Dec 2019   Prob (F-statistic):           1.94e-78
Time:                        21:49:54   Log-Likelihood:                 280.10
No. Observations:                 284   AIC:                            -542.2
Df Residuals:                     275   BIC:                            -509.4
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
MEI            0.0642      0.006      9.923      0.000       0.051       0.077
CO2            0.0065      0.002      2.826      0.005       0.002       0.011
CH4            0.0001      0.001      0.240      0.810      -0.001       0.001
N2O           -0.0165      0.009     -1.930      0.055      -0.033       0.000
CFC-11        -0.0066      0.002     -4.078      0.000      -0.010      -0.003
CFC-12         0.0038      0.001      3.757      0.000       0.002       0.006
TSI            0.0931      0.015      6.313      0.000       0.064       0.122
Aerosols      -1.5376      0.213     -7.210      0.000      -1.957      -1.118
const       -124.5943     19.887     -6.265      0.000    -163.744     -85.445
==============================================================================
Omnibus:                        8.740   Durbin-Watson:                   0.956
Prob(Omnibus):                  0.013   Jarque-Bera (JB):               10.327
Skew:                           0.289   Prob(JB):                      0.00572
Kurtosis:                       3.733   Cond. No.                     8.53e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.53e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
[Finished in 2.8s]
```

Therefore, MEI, CO2, CFC-11, CFC-12, TSI, Aerosols are significant at 1%, as their p-value is less than 0.005.

(4)

The independent variables should not be correlated and thus closed form solution $X^T X$ is invertible.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns

print "Problem 1"
df = pd.read_csv('climate_change_2.csv')
df['const']=1
cols = list(df)
cols.insert(11, cols.pop(cols.index('const')))
df = df.loc[:,cols]
cols = list(df)
Training_set = df.loc[df['Year'] <= 2006]
Testing_set = df.loc[df['Year'] > 2006]
X = np.matrix(Training_set.iloc[:, 2:12].values)
Y = np.matrix(Training_set.iloc[:, 12].values).T
X_test = np.matrix(Testing_set.iloc[:, 2:12].values)
Y_test = np.matrix(Testing_set.iloc[:, 12].values)

def closed_form_1(X, Y):
    Theta = np.dot(np.dot(np.dot(X.T, X).I, X.T), Y)
    return Theta

Theta = closed_form_1(X, Y)
temp_est = np.dot(X, Theta)
temp_est_test = np.dot(X_test, Theta)


def R_square(temp_est, Y):
    var_xb = (temp_est - temp_est.mean()).var()
    var_y = (Y - temp_est.mean()).var()
    R2 = var_xb / var_y
    return R2

R2 = R_square(temp_est,Y)
R2_test = R_square(temp_est_test, Y_test)

result = (np.linalg.matrix_rank(np.dot(X.T,X)) < np.dot(X.T,X).shape[0]) & (np.linalg.matrix_rank(np.dot(X.T,X)) < np.dot(X.T,X).shape[1])
print result
```

```
Problem 1
True
[Finished in 2.4s]
```

Then its rank is less than shape (both rows and columns), so it is invertible.

And another reason is that it is likely to get the coefficients to be positive, however, some are negative, there must be some correlations and may link to the invertibility.

Therefore, the solution is unreasonable.

Problem 2:

(1)

L1:

$$Loss\ Function = \frac{1}{2}(X\theta - Y)^T(X\theta - Y) + \alpha|\theta|$$

L2:

$$Loss\ Function = \frac{1}{2}(X\theta - Y)^T(X\theta - Y) + \frac{1}{2}\alpha\theta^2$$

(2)

```
lamb = 0.5
def closed_form_2(X, Y, lamb):
    Theta2 = np.dot(np.dot((np.dot(X.T, X) + lamb * np.eye(X.shape[1])).I, X.T), Y)
    return Theta2

Theta2 = closed_form_2(X, Y, lamb)
temp_est2 = np.dot(X, Theta2)
temp_est_test2 = np.dot(X_test, Theta2)
R2_2 = R_square(temp_est2, Y)
R2_test_2 = R_square(temp_est_test2, Y_test)
print "R2_train:", R2_2
print "R2_test:", R2_test_2
```

```
Problem 2
R2_train: 0.6815168002731263
R2_test: 0.11276203841963961
[Finished in 12.6s]
```

(3)

In the theory, L2 regularization decreases the unnecessary coefficients to almost 0 and remains the necessary ones and thus let the necessary ones more significant.

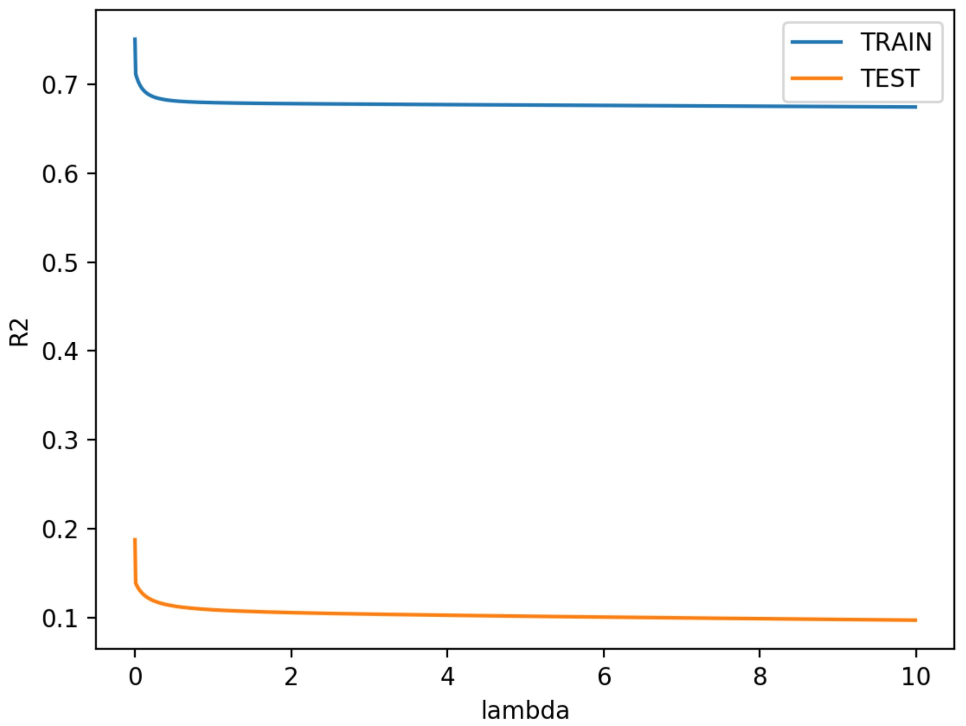
```
def p_value(X, Y, Theta):
    error = np.sum(np.square(np.dot(X,Theta)-Y))
    cii = (np.dot(X.T,X)).I.diagonal()
    sigma2 = error /(X.shape[0]-X.shape[1])
    S=np.sqrt(sigma2*cii)
    results = pd.DataFrame()
    results['index'] = df.columns.values[2:11]
    results.set_index(['index'], inplace=True)
    results['coefficient'] = Theta
    t =Theta.T / S
    results['t-statistics']=np.array(t)[0]
    p = stats.t.sf(np.abs(t), (X.shape[0]-X.shape[1]))*2
    results['p-value']=np.array(p)[0]
    return results

print p_value(X, Y, Theta)
```

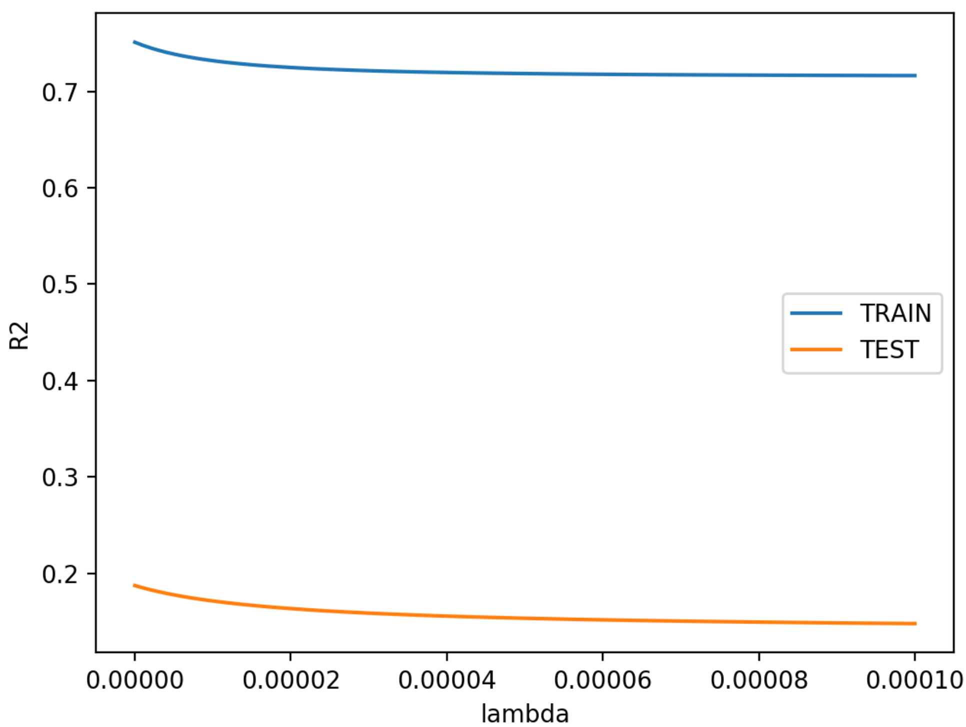
```
Problem 2
R2_train: 0.6815168002731263
R2_test: 0.11276203841963961
          coefficient  t-statistics      p-value
index
MEI          0.056840      8.387262  2.628394e-15
CO2          0.006280      2.624397  9.165479e-03
CH4          0.000080      0.148189  8.823022e-01
N2O         -0.013822     -1.540741  1.245298e-01
CFC-11      -0.005941     -3.488382  5.656630e-04
CFC-12       0.003619      3.408916  7.499122e-04
TSI          0.017357      1.123112  2.623694e-01
Aerosols    -1.366851     -6.119434  3.221969e-09
const      -21.849072     -1.048943  2.951252e-01
[Finished in 4.4s]
```

We find that p_values are significantly lower than before and that means using L2 regularization is more robust.

(4)

```
print "Problem 2"

lamb = 0.5
def closed_form_2(X, Y, lamb):
    Theta2 = np.dot(np.dot((np.dot(X.T, X) + lamb * np.eye(X.shape[1])).I, X.T), Y)
    return Theta2

Theta2 = closed_form_2(X, Y, lamb)
temp_est2 = np.dot(X, Theta2)
temp_est_test2 = np.dot(X_test, Theta2)
R2_2 = R_square(temp_est2, Y)
R2_test_2 = R_square(temp_est_test2, Y_test)
print "R2_train:", R2_2
print "R2_test:", R2_test_2

lamb_list = np.arange(0, 10, 0.01)
Theta2_list = []
R2_2_list = []
R2_test_2_list = []
for i in lamb_list:
    Theta2 = closed_form_2(X, Y, i)
    Theta2_list.append(Theta2)
    R2_2_list.append(R_square(np.dot(X,Theta2), Y))
    R2_test_2_list.append(R_square(np.dot(X_test,Theta2), Y_test))
'''
print len(Theta2_list)
print len(R2_2_list)
print len(R2_test_2_list)
'''
plt.plot(lamb_list, R2_2_list, label = 'TRAIN')
plt.plot(lamb_list, R2_test_2_list, label = 'TEST')
plt.xlabel('lambda')
plt.ylabel('R2')
plt.legend()
plt.show()

### Then the best lambda must be <1

### After so many tests

lamb_list = np.arange(0, 0.0001, 0.000001)
Theta2_list = []
R2_2_list = []
R2_test_2_list = []
for i in lamb_list:
    Theta2 = closed_form_2(X, Y, i)
    Theta2_list.append(Theta2)
    R2_2_list.append(R_square(np.dot(X,Theta2), Y))
    R2_test_2_list.append(R_square(np.dot(X_test,Theta2), Y_test))
'''
```


TRAIN
TEST
R2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2
4
6
8
10
lambda


TRAIN
TEST
R2
0.2
0.3
0.4
0.5
0.6
0.7
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
lambda

Problem 3:

(1)

```
print "Problem 3"

corr = df.iloc[:,2:10].corr()
f, ax = plt.subplots(figsize=(12, 10))
print(corr)
print(corr > 0.8)
```

```
Problem 3
                MEI       CO2       CH4  ...    CFC-12       TSI  Aerosols
MEI        1.000000 -0.152911 -0.105555  ... -0.039836 -0.076826  0.352351
CO2       -0.152911  1.000000  0.872253  ...  0.823210  0.017867 -0.369265
CH4       -0.105555  0.872253  1.000000  ...  0.958237  0.146335 -0.290381
N2O       -0.162375  0.981135  0.894409  ...  0.839295  0.039892 -0.353499
CFC-11     0.088171  0.401284  0.713504  ...  0.831381  0.284629 -0.032302
CFC-12    -0.039836  0.823210  0.958237  ...  1.000000  0.189270 -0.243785
TSI       -0.076826  0.017867  0.146335  ...  0.189270  1.000000  0.083238
Aerosols   0.352351 -0.369265 -0.290381  ... -0.243785  0.083238  1.000000

[8 rows x 8 columns]
             MEI    CO2    CH4    N2O  CFC-11  CFC-12    TSI  Aerosols
MEI         True  False  False  False   False   False  False     False
CO2        False   True   True   True   False    True  False     False
CH4        False   True   True   True   False    True  False     False
N2O        False   True   True   True   False    True  False     False
CFC-11     False  False  False  False    True    True  False     False
CFC-12     False   True   True   True    True    True  False     False
TSI        False  False  False  False   False   False   True     False
Aerosols   False  False  False  False   False   False  False      True
[Finished in 5.0s]
```

We can find that CO2, CH4, N2O and CFC-12 are highly correlated, we can drop three of them, for example, CO2, CH4 and N2O.

(2)

```
print "Problem 3"

corr = df.iloc[:,2:10].corr()
f, ax = plt.subplots(figsize=(12, 10))
print(corr)
print(corr > 0.8)

new_df = df.copy()
new_df.drop(['CO2','CH4','N2O'],axis=1,inplace=True)
new_Training_set = new_df.loc[new_df['Year']<=2006]
new_Testing_set = new_df.loc[new_df['Year']>2006]
new_X = np.matrix(Training_set.iloc[:,2:11].values)
new_Y = np.matrix(Training_set.iloc[:,11].values).T
new_X_test = np.matrix(Testing_set.iloc[:,2:11].values)
new_Y_test = np.matrix(Testing_set.iloc[:,11].values).T

new_Theta = closed_form_2(new_X,new_Y,lamb)

print R_square(np.dot(new_X, new_Theta), new_Y)
print R_square(np.dot(new_X_test, new_Theta), new_Y_test)
```

```
Problem 3
               MEI       CO2       CH4  ...    CFC-12       TSI  Aerosols
MEI       1.000000 -0.152911 -0.105555  ... -0.039836 -0.076826  0.352351
CO2      -0.152911  1.000000  0.872253  ...  0.823210  0.017867 -0.369265
CH4      -0.105555  0.872253  1.000000  ...  0.958237  0.146335 -0.290381
N2O      -0.162375  0.981135  0.894409  ...  0.839295  0.039892 -0.353499
CFC-11    0.088171  0.401284  0.713504  ...  0.831381  0.284629 -0.032302
CFC-12   -0.039836  0.823210  0.958237  ...  1.000000  0.189270 -0.243785
TSI      -0.076826  0.017867  0.146335  ...  0.189270  1.000000  0.083238
Aerosols  0.352351 -0.369265 -0.290381  ... -0.243785  0.083238  1.000000

[8 rows x 8 columns]
            MEI    CO2    CH4    N2O  CFC-11  CFC-12    TSI  Aerosols
MEI        True  False  False  False   False   False  False     False
CO2       False   True   True   True   False    True  False     False
CH4       False   True   True   True   False    True  False     False
N2O       False   True   True   True   False    True  False     False
CFC-11    False  False  False  False    True    True  False     False
CFC-12    False   True   True   True    True    True  False     False
TSI       False  False  False  False   False   False   True     False
Aerosols  False  False  False  False   False   False  False      True
0.694468408539303
0.12660600610369788
[Finished in 2.7s]
```

Problem 4:

```python
def Gradient_Descent(alpha, theta_0, x, y, tol):
    theta = theta_0
    cost = (1./len(x)) * (x.T @ (x @ theta_0 - y))
    count = 0
    while not ((np.all(cost) <= tol) | (count > 10000)):
        count += 1
        theta = theta - alpha * (1./len(x)) * (x.T @ (x @ theta - y))
        cost = (1./len(x)) * (x.T @ (x @ theta - y))
    print('iterate {} times'.format(count))
    return theta
```