Big Data-Climate Change

1. **Problem one—Creating Your First Model**
2. Implement a function closed\_form\_1 that computes this closed form solution given the features X, labels Y(using Python or Matlab)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import statsmodels.api as sm

df = pd.read\_csv('climate\_change\_1.csv')

def closed\_form\_1():

traindata = df[df.Year<=2006]

testdata = df[df.Year>2006]

X = traindata[['MEI','CO2','CH4','N2O','CFC-11','CFC-12','TSI','Aerosols']]

Y = traindata['Temp']

est = sm.OLS(Y, sm.add\_constant(X)).fit()

print(est.summary())

print(est.params)

closed\_form\_1()

1. Write down the mathematical formula for the linear model and evaluate the model on the training set and the testing set.

The on training set is 0.744

The on testing set is 0.425

1. Which variables are significant in the model?

If the p-value is below 0.05, we will choose MEI, CO2, CFC-11, CFC-12, TSI, Aerosols.

1. Write down the necessary conditions for using the closed form solution. And you can apply it to the dataset climate\_change\_2.csv, explain the solution is unreasonable.

Necessary conditions: MEI, CO2, CH4, N2O, CFC-11, CFC-12, TSI, Aerosols should not be correlated with each other.

Unreasonable: N2O and CFC-11 are greenhouse gases and the regression coefficients of them should be positive but the results are negative. Therefore, N2O and CFC-11 are correlated with other variables in the data set.

1. **Problem two—Regularization**

**Regularization is a method to boost robustness of model, including regularization and regularization.**

1. **Please write down the loss function for linear model with regularization and regularization, respectively.**

Regularization:

Regularization:

represents the original cost function.

1. **The closed form solution for linear model with regularization.**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import statsmodels.api as sm

df = pd.read\_csv('climate\_change\_1.csv')

def closed\_form\_2():

lamb = 0.5

df['const'] = 1

traindata = df[df.Year<=2006]

testdata = df[df.Year>2006]

X = traindata[['MEI','CO2','CH4','N2O','CFC-11','CFC-12','TSI','Aerosols','const']]

Y = traindata['Temp']

Theta2 = np.dot(

np.dot(np.linalg.inv((np.dot(X.T, X) + lamb \* np.eye(X.shape[1]))), X.T), Y)

print(Theta2)

closed\_form\_2()

**Result:**

|  |  |
| --- | --- |
| Item |  |
| MEI | 4.55768014e-02 |
| CO2 | 7.80443532e-03 |
| CH4 | 1.95701031e-04 |
| N2O | -1.64893727e-02 |
| CFC-11 | -6.38359095e-03 |
| CFC-12 | 3.74766007e-03 |
| TSI | 1.44919104e-03 |
| Aerosols | -3.65599605e-01 |
| Const | -4.68953239e-03 |

1. **Compare the two solutions in problem 1 and problem 2 and explain the reason why linear model with regularization is robust.**

The reason why the method is better than before is that the size of the coefficient of the constant term of the method is basically the same as the value of the temp result, which will not cause a large deviation of the result due to the error of the constant term estimation.

1. **You can change the regularization parameter λ to get different solutions for this problem.**

|  |  |  |
| --- | --- | --- |
|  | **of Training Set** | **of Testing Set** |
| **10** | **0.6746079231515448** | **0.9408716921954439** |
| **1** | **0.6794692110104662** | **0.8467501178134881** |
| **0.1** | **0.6944684109361836** | **0.6732879125422909** |
| **0.01** | **0.7116529617244923** | **0.5852763146622774** |
| **0.001** | **0.7148330433597858** | **0.5625217958135218** |

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import statsmodels.api as sm

df = pd.read\_csv('climate\_change\_1.csv')

def R\_square(Y\_pred, Y):

ESS = np.sum((Y\_pred - Y.mean())\*\*2)

TSS = np.sum((Y - Y.mean())\*\*2)

R2 = ESS / TSS

print(R2)

def closed\_form\_2():

lamb = 0.001

df['const'] = 1

traindata = df[df.Year<=2006]

testdata = df[df.Year>2006]

X = traindata[['MEI','CO2','CH4','N2O','CFC-11','CFC-12','TSI','Aerosols','const']]

Y = traindata['Temp']

Theta2 = np.dot(

np.dot(np.linalg.inv((np.dot(X.T, X) + lamb \* np.eye(X.shape[1]))), X.T), Y)

print(Theta2)

Y\_pred = np.dot(X,Theta2)

R\_square(Y\_pred,Y)

closed\_form\_2()

1. **Problem three—Feature Selection**
2. **From Problem 1， you can know which variables are significant, therefore you can use less variables to train model. For example, remove highly correlated and redundant features. You can propose a workflow to select feature.**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import statsmodels.api as sm

df = pd.read\_csv('climate\_change\_1.csv')

climate\_change\_1\_train = df[df.Year<=2006]

climate\_change\_1\_test = df[df.Year>2006]

climate\_change\_1\_train\_X = climate\_change\_1\_train.iloc[:,2:10]

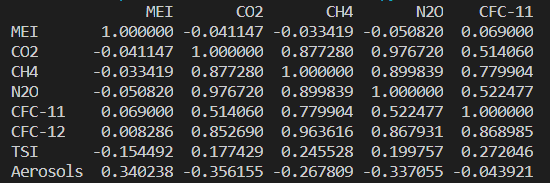
climate\_change\_1\_train\_Y = climate\_change\_1\_train.iloc[:,10]

print(climate\_change\_1\_train\_X.corr())

#80% above is considered as highly related

#CO2 CH4 N2O CFC-12 are highly correlated with each other

#CFC-11 CFC-12 are correlated with each other



#delete N2O,CH4,CFC-12

climate\_change\_1\_train\_X\_new = climate\_change\_1\_train.loc[:,['MEI','CO2','CFC-11','TSI','Aerosols']]

# 1.2 run a regression remove not significant with alpha = 0.01

import statsmodels.api as sm

from statsmodels import regression

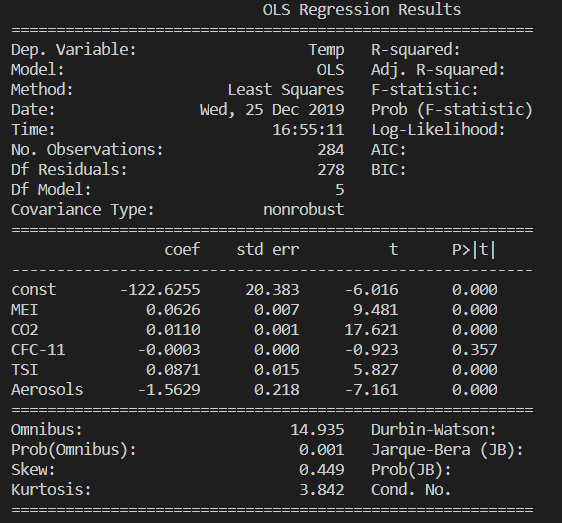
climate\_change\_1\_train\_X\_new=sm.add\_constant(climate\_change\_1\_train\_X\_new)

model=sm.OLS(climate\_change\_1\_train\_Y,climate\_change\_1\_train\_X\_new)

res=model.fit()

print(res.summary())

#delete CFC-11 which is not significant



# 1.3 run the final model

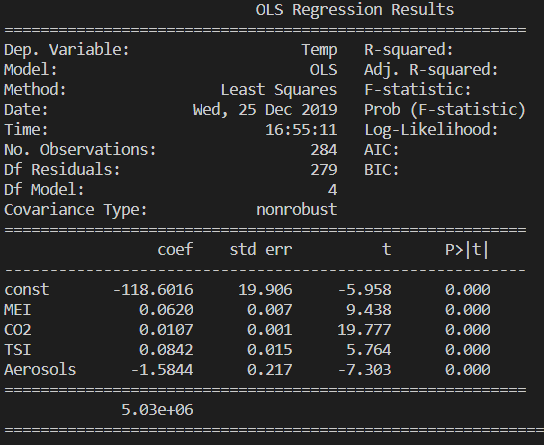
climate\_change\_1\_train\_X\_new\_2 = climate\_change\_1\_train.loc[:,['MEI','CO2','TSI','Aerosols']]

climate\_change\_1\_train\_X\_new\_2=sm.add\_constant(climate\_change\_1\_train\_X\_new\_2)

model=sm.OLS(climate\_change\_1\_train\_Y,climate\_change\_1\_train\_X\_new\_2)

res=model.fit()

print(res.summary())



Result:

We will use MEI, CO2, TSI and Aerosols.

1. **Train a better model than the model in Problem 2.**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import statsmodels.api as sm

df = pd.read\_csv('climate\_change\_1.csv')

def R\_square(Y\_pred, Y):

ESS = np.sum((Y\_pred - Y.mean())\*\*2)

TSS = np.sum((Y - Y.mean())\*\*2)

R2 = ESS / TSS

print(R2)

def closed\_form\_2():

lamb = 0.1

df['const'] = 1

traindata = df[df.Year<=2006]

testdata = df[df.Year>2006]

X = traindata[['MEI','CO2','TSI','Aerosols',’const’]]

Y = traindata['Temp']

Theta2 = np.dot(

np.dot(np.linalg.inv((np.dot(X.T, X) + lamb \* np.eye(X.shape[1]))), X.T), Y)

print(Theta2)

Y\_pred = np.dot(X,Theta2)

R\_square(Y\_pred,Y)

closed\_form\_2()

1. **Problem Four-Gradient Descent**

**Iterative Expression:**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import statsmodels.api as sm

from sklearn.preprocessing import normalize

df = pd.read\_csv('climate\_change\_1.csv')

def Gradient\_Descent(alpha, theta\_0, x, y, tol):

theta = theta\_0

cost = (1./(2\*len(x))) \* (np.sum((x @ theta\_0 - y)\*\*2))

count = 0

while not ((cost <= tol) | (count > 10000)):

count += 1

theta = theta - alpha \* (1./len(x)) \* (x.T @ (x @ theta - y))

cost = (1./(2\*len(x))) \* (np.sum((x @ theta\_0 - y)\*\*2))

print(theta)

print('迭代共{}次'.format(count))

print(theta)

def regularit(df):

newDataFrame = pd.DataFrame(index=df.index)

columns = df.columns.tolist()

for c in columns:

if c != 'const':

d = df[c]

MAX = d.max()

MIN = d.min()

newDataFrame[c] = ((d - MIN) / (MAX - MIN)).tolist()

else:

newDataFrame[c] = 1

return newDataFrame

def closed\_form\_2():

lamb = 0.1

df['const'] = 1

traindata = df[df.Year<=2006]

testdata = df[df.Year>2006]

X = traindata[['MEI','CO2','CH4','N2O','CFC-11','CFC-12','TSI','Aerosols','const']]

Y = traindata['Temp']

X = regularit(X)

Y = (Y-Y.mean())/(Y.max()-Y.min())

print(X)

print(Y)

Theta2 = [1,1,1,1,1,1,1,1,1]

Gradient\_Descent(0.001, Theta2, X, Y, 10)

closed\_form\_2()