Homework 2——Big Data Analysis

Name: 廖家瑞 (Jiarui Liao)   Student ID: 1801212884   Class: Fintech

1 Problem 1——Creating Your First Model

1.1 Solution:

```
import pandas as pd
import numpy as np
# Read the dataset
raw_df=pd.read_csv('climate_change_1.csv')
print(raw_df)
# Split the data into training set
X_test=raw_df[raw_df['Year']>2006].iloc[:,:-1]
y_test=raw_df[raw_df['Year']>2006].iloc[:,-1]
X_train=raw_df[raw_df['Year']<=2006].iloc[:,:-1]
y_train=raw_df[raw_df['Year']<=2006].iloc[:,-1]
print(X_test)
# computes the closed form solution
def closed_form(X_train,y_train):
    return np.dot(np.matmul(np.linalg.inv(np.matmul(X_train.transpose(),X_train)),X_train.transpose()),y_train)
theta=closed_form(np.array(X_train),np.array(y_train))
print(theta)
```

```
[ -4.24887417e-02  -6.16714875e-03   6.48363535e-02   6.46994231e-03
  -1.57088223e-04   2.29289621e-02  -1.00275979e-02   6.77771348e-03
   5.49139595e-02  -1.64648727e+00]
```

1.2 Solution:

```
#Write down the mathematical formula for the linear model
Y_fit = np.dot (X_train, theta)
```

#Evaluate the model R^2

```
y_fit=np.dot(X_train,theta)
y_mean=np.mean(y_train)
print("R_Square of training set:")
print(np.dot(y_fit-y_mean,y_fit-y_mean)/np.dot(np.array(y_train)-y_mean,np.array(y_train)-y_mean))
```

```
R_Square of training set:
0.7489870281004234
```

```
y_test_fit=np.dot(X_test,theta)
y_test_mean=np.mean(y_test)
print("R_Square of test set:")
print(np.dot(y_test_fit-y_test_mean,y_test_fit-y_test_mean)/np.dot(np.array(y_test)-y_test_mean,np.array(y_test)-
```

y_test_mean))

```
R_Square of test set:
0.261613883646
```

1.3 Solution:

```
Theta = [ -4.24887417e-02  -6.16714875e-03   6.48363535e-02   6.46994231e-03
  -1.57088223e-04   2.29289621e-02  -1.00275979e-02   6.77771348e-03
   5.49139595e-02  -1.64648727e+00]
```

The larger theta is, the more significant the corresponding variable is.

The most significant variable: `Aerosols\Year\MEI\N2O\CFC-11\TSI`

1.4 Solution:

The necessary conditions for using the closed form solution:

1) X full rank, X^TX invertible
2) $E[\varepsilon|X] = 0$
3) Var $[\varepsilon|X] = \sigma^2 I$

Why the solution is unreasonable:

Because Some independent variables have autocorrelation problems.

2 Problem 2——Regularization

2.1 Solution:

L1 regularization:

$$\min \frac{1}{2m}\sum_{i=1}^{m}(f(x)-y^{(i)})^2 + C||\omega||_1$$

L2 regularization:

$$\min \frac{1}{2m}\sum_{i=1}^{m}(f(x)-y^{(i)})^2 + C||\omega||_2^2$$

2.2 Solution:

```
def closed_form_2(X_train,y_train,Lambda):
    return np.dot(np.matmul(np.linalg.inv(np.matmul(X_train.transpose(),X_train)-Lambda*np.identity(X_train.shape[1])),X_train.transpose()),y_train)
Lambda=1
theta2=closed_form_2(np.array(X_train),np.array(y_train),Lambda)
print(theta2)
```

```
[ -2.68848686e-02  -4.40486101e-03   4.27538087e-02   9.06507355e-03
   1.77964222e-04   3.86582615e-03  -9.70260899e-03   5.97524113e-03
   3.55660849e-02   3.59229422e-01]
```

2.3 Solution:

```
#Compare the two solutions in problem 1 and problem 2:
y_fit=np.dot(X_train,theta2)
y_mean=np.mean(y_train)
print("R_Square of training set:")
print(np.dot(y_fit-y_mean,y_fit-y_mean)/np.dot(np.array(y_train)-y_mean,np.array(y_train)-y_mean))
# print(np.array(y_train))
```

```
R_Square of training set:
0.700274534986
```

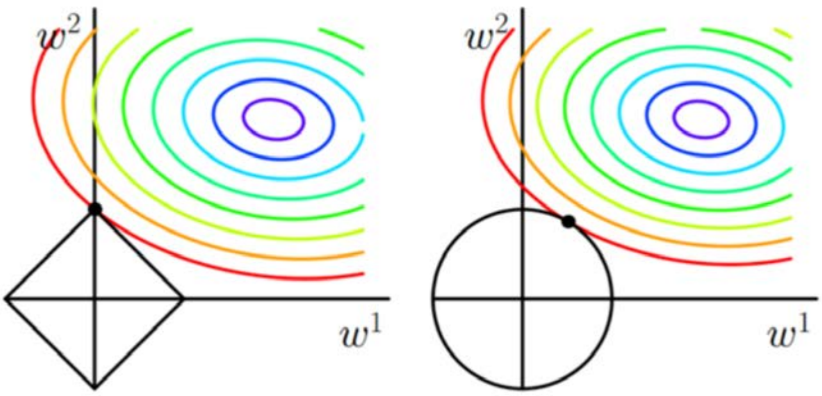
```
y_test_fit=np.dot(X_test,theta2)
y_test_mean=np.mean(y_test)
# print(y_average)
print("R_Square of test set:")
print(np.dot(y_test_fit-y_test_mean,y_test_fit-y_test_mean)/np.dot(np.array(y_test)-y_test_mean,np.array(y_test)-
y_test_mean))
```

```
R_Square of test set:
0.464645125728
```

R_Square of test set (Problem 2) > R_Square of test set (Problem 1)

#why linear model with L2 regularization is robust:

When the loss function of L2 regularization is minimal, all parameters are usually the same, and each variable is given the same weight without emphasizing a certain variable. Considering the global characteristics, it has robustness, while L1 has sparsity.



(a) ℓ_1**-ball meets quadratic function.** ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2**-ball meets quadratic function.** ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes.

2.4 Solution:

```
Lambda_list = [0.001,0.01,0.1,1,10]
for Lambda in Lambda_list:
    theta2=closed_form_2(np.array(X_train),np.array(y_train),Lambda)
    y_fit=np.dot(X_train,theta2)
```

```
    y_mean=np.mean(y_train)
    print("Lambda=", Lambda)
    print()
    print("R_Square of training set:", np.dot(y_fit-y_mean,y_fit-y_mean)/np.dot(np.array(y_train)-y_mean,np.array(y_train)-y_mean))
    print(pd.DataFrame([y_fit,y_train]))

    y_test_fit=np.dot(X_test,theta2)
    y_test_mean=np.mean(y_test)
    print("R_Square of test set:", np.dot(y_test_fit-y_test_mean,y_test_fit-y_test_mean)/np.dot(np.array(y_test)-y_test_mean,np.array(y_test)-y_test_mean))
```

```
Lambda= 0.001

R_Square of training set: 0.7495780710535302
        0         1         2         3         4        5         6   \
0  0.15375  0.138685  0.121763  0.087065  0.023763 -0.00764 -0.001793
1  0.10900  0.118000  0.137000  0.176000  0.149000  0.09300  0.232000

         7         8         9    ...       274       275       276       277  \
0  0.010186  0.004465  0.009832  ...  0.394054  0.392035  0.437296  0.461777
1  0.078000  0.089000  0.013000  ...  0.380000  0.378000  0.352000  0.442000

        278       279       280       281       282       283
0  0.451102  0.438407  0.432663  0.434633  0.458168  0.442598
1  0.456000  0.482000  0.425000  0.472000  0.440000  0.518000

[2 rows x 284 columns]
R_Square of test set: 0.2623254047426737
Lambda= 0.01

R_Square of training set: 0.7553790551048112
        0         1         2         3         4         5         6   \
0  0.150473  0.135805  0.119347  0.084935  0.021256 -0.010494 -0.004415
1  0.109000  0.118000  0.137000  0.176000  0.149000  0.093000  0.232000

         7         8         9    ...       274       275       276       277  \
0  0.007752  0.001656  0.007279  ...  0.392181  0.389482  0.435458  0.460739
1  0.078000  0.089000  0.013000  ...  0.380000  0.378000  0.352000  0.442000

        278       279       280       281       282       283
0  0.450384  0.437928  0.432281  0.434126  0.457837  0.441694
1  0.456000  0.482000  0.425000  0.472000  0.440000  0.518000
```

```
[2 rows x 284 columns]
R_Square of test set: 0.26989557865532154
Lambda= 0.1

R_Square of training set: 0.9668693311144463
         0         1         2         3         4         5         6  \
0  0.07623  0.070551  0.064583  0.036665 -0.035477 -0.075061 -0.063737
1  0.10900  0.118000  0.137000  0.176000  0.149000  0.093000  0.232000

          7         8        9  ...       274       275      276       277  \
0 -0.047345 -0.061831 -0.05043  ...  0.349914  0.331802  0.39394  0.437306
1  0.078000  0.089000  0.01300  ...  0.380000  0.378000  0.35200  0.442000

        278       279       280       281       282       283
0  0.434195  0.427146  0.423692  0.422701  0.450378  0.421284
1  0.456000  0.482000  0.425000  0.472000  0.440000  0.518000

[2 rows x 284 columns]
R_Square of test set: 0.8200829029899867
Lambda= 1

R_Square of training set: 0.7002745349237837
          0         1         2         3         4         5         6  \
0  0.229869  0.205673  0.178147  0.136752  0.081454  0.057668  0.058342
1  0.109000  0.118000  0.137000  0.176000  0.149000  0.093000  0.232000

         7         8         9  ...       274       275       276       277  \
0  0.06616  0.067994  0.067741  ...  0.435917  0.449449  0.478746  0.485091
1  0.07800  0.089000  0.013000  ...  0.380000  0.378000  0.352000  0.442000

        278       279      280       281       282       283
0  0.467036  0.448734  0.44079  0.445645  0.465325  0.462824
1  0.456000  0.482000  0.42500  0.472000  0.440000  0.518000

[2 rows x 284 columns]
R_Square of test set: 0.46464512531827645
Lambda= 10

R_Square of training set: 0.7097751568666681
          0         1         2         3         4         5         6  \
0  0.230692  0.207313  0.181261  0.139584  0.077776  0.049918  0.052629
1  0.109000  0.118000  0.137000  0.176000  0.149000  0.093000  0.232000

          7         8         9  ...       274       275       276       277  \
```

```
0  0.062035  0.053445  0.056232  ...  0.422326  0.433451  0.468355  0.478405
1  0.078000  0.089000  0.013000  ...  0.380000  0.378000  0.352000  0.442000

        278       279       280       281       282       283
0  0.460765  0.441737  0.434216  0.438785  0.460393  0.456114
1  0.456000  0.482000  0.425000  0.472000  0.440000  0.518000

[2 rows x 284 columns]
R_Square of test set: 0.31572868589018216
```

Lambda= 0.1 is the best regularization parameter Lambda.

3 Problem 3——Feature Selection

3.1 Solution：

We can use PCA to reduce dimension. Principal Component Analysis (PCA) is the most widely used data dimension reduction algorithm.
Specific algorithm work flow:

1) Set M n-dimensional data:
2) Form the raw data into N row M column matrix X in columns
3) We're going to zero mean every row of X, so we're going to subtract the mean of every row
4) Find the covariance matrix C for X
5) Find the eigenvalues of the covariance matrix C and the corresponding eigenvectors. The eigenvalues of C are the variances of each dimension of Y and the diagonal elements of D.
6) The eigenvectors are arranged into matrices from top to bottom according to the corresponding eigenvalues. According to the actual business scenario, the first R rows are taken to form the matrix P
7) Y=PX is the target matrix after R dimension is reduced

3.2 Solution：

```
from sklearn.decomposition import PCA
import numpy as np
from sklearn.preprocessing import StandardScaler
# feature normalization (feature scaling)
X_scaler = StandardScaler()
x = X_scaler.fit_transform(X_train)

# PCA: Keep 80% of the information after dimension reduction
pca = PCA(n_components=0.8)
pca.fit(x)
pca.transform(x)
```

4 Problem 4——Gradient Descent

```
def normalization(X_train):
    mean=np.mean(X_train)
    M1=np.max(X_train)
    M2=np.min(X_train)
    X=(X_train-mean)/(M1-M2)
    return X,mean,M1,M2

X_train_norm,mean_train,max_train,min_train=normalization(X_train)
def gradient_Decent(X_train,y_train,Lambda,alpha):
        iters=10000
        X=X_train.copy()
        X.insert(0,'intercept',np.ones(len(X)))
        theta=np.zeros(X.shape[1])
        delta=np.matmul(X.T,np.matmul(X,theta)-y_train)

        for i in range(iters):
                delta=np.matmul(X.T,np.matmul(X,theta)-y_train)
                theta[0]=theta[0]-(alpha/X.shape[0])*(delta[0])
                theta[1:]=theta[1:]-(alpha/X.shape[0])*(delta[1:]+Lambda*theta[1:])

        return theta
for Lambda in Lambda_list:

        theta2=gradient_decent(X_train_norm,y_train,Lambda,0.1)
        y_fit=theta2[0]+np.matmul(X_train_norm,theta2[1:])
        y_mean=np.mean(y_train)
        print("Lambda=",end=" ")
        print(Lambda)
        print("R_Square of training set:",end=" ")
        print(1-np.matmul(y_fit-y_train,y_fit-y_train)/np.matmul(y_train-y_mean,y_train-y_mean),end=", ")
        y_test_fit=theta2[0]+np.matmul((X_test-mean_train)/(max_train-min_train),theta2[1:])
        y_test_mean=np.mean(y_test)
        print("R_Square of test set:", 1-np.matmul(y_test_fit-y_test,y_test_fit-y_test)/np.matmul(y_test-
y_test_mean,y_test-y_test_mean))
```

```
lamda= 0.001
R_Square of training set: 0.750805899358, R_Square of test set: -0.0289549772534
lamda= 0.01
R_Square of training set: 0.750732356691, R_Square of test set: -0.0324290719715
lamda= 0.1
R_Square of training set: 0.750017309963, R_Square of test set: -0.0651262903629
lamda= 1
```

```
R_Square of training set: 0.744364431705, R_Square of test set: -0.256411387255
lamda= 10
R_Square of training set: 0.692707642976, R_Square of test set: -0.424000104237
```