The codes and explanations are reasonable.

CA1

February 28, 2020

1 Compute Assignment 1

2 part a)

Let us define $L = \frac{1}{N} \sum_{i \in [N]} ||w^T x_i - y_i||^2 + \lambda ||w||^2$

Then

$$\frac{\partial L}{\partial w} = \frac{1}{N} \sum_{i \in [N]} \frac{\partial x_i^T w w^T x_i}{\partial w} - 2 \frac{\partial y_i^T w^T x_i}{\partial w} + 2\lambda w \tag{1}$$

$$= \frac{1}{N} \sum_{i \in [N]} 2 x_i x_i^T w - 2 x_i y_i^T + 2\lambda w \tag{2}$$

$$= 0, \tag{3}$$

so

$$\frac{1}{N} \sum_{i \in [N]} x_i y_i^T = \frac{1}{N} \sum_{i \in [N]} x_i x_i^T w + \lambda w \tag{4}$$

So

$$w^* = (X X^T + \lambda I)^{-1} X Y^T$$

where $X = [x_1, x_2, \ldots, x_N]$ and $Y = [y_1, y_2, \ldots, y_N]$. Note that this holds if X is a fat matrix. On the other hand, if X is a tall matrix we should compute the right inverse as follows:

$$w^* = X(X^T X + \lambda I)^{-1} Y^T \quad \text{w*=(X\^T X + \lambda N I)\^-1 X\^T Y}$$

[29]:
```
##imports from libraries
import pandas as pd
import numpy as np
import time
from sklearn import linear_model
```

[9]:
```
## Load data and preprocessing

## Example of reading dataset ''crimedata.csv'' :
data = pd.read_csv("household_power_consumption.txt", sep=";")
Y = data.iloc[:,2:4]
```

```
X = data.iloc[:,4:8]
## You can observe the shape of data by
print(X.shape)
print(Y.shape)

X_ = np.array(X.replace({'?':0})).T
Y_ = np.array(Y.replace({'?':0})).T

X_ = X_.astype(float)
Y_ = Y_.astype(float)

#Y_ = np.array(Y).reshape(Y.shape[1],Y.shape[0])
```

```
(2075259, 4)
(2075259, 2)
[[234.84 233.63 233.29 ... 239.82 239.7  239.55]
 [ 18.4   23.    23.   ...   3.8    3.8    3.8 ]
 [  0.     0.     0.   ...   0.     0.     0.  ]
 [  1.     1.     2.   ...   0.     0.     0.  ]]
[[4.216 5.36  5.374 ... 0.938 0.934 0.932]
 [0.418 0.436 0.498 ... 0.    0.    0.   ]]
```

[32]:
```
## Closed form solution and optimal linear regressor

# Define lambda here:
lam = 0 # change the value

start = time.time()
## Calculate the closed-form solution here:
A=X_.dot(X_.T)+lam*np.eye(X_.shape[0])
w=np.linalg.inv(A).dot(X_).dot(Y_.T)
end = time.time()
print('my result')
print('time=',end-start,' seconds')
print('w=\n',w)
print('MSE',np.linalg.norm(Y_-w.T.dot(X_),ord='fro')/X_.shape[1],'\n\n')


reg = linear_model.Ridge(alpha=0)
start = time.time()
## Find the optimal linear regressor here:
reg.fit(X_.T,Y_.T)
end = time.time()
w_sk=reg.coef_.T
print('sklearn:')
print('time=',end-start,' seconds')
print('w=\n',w_sk)
```

```
print('MSE',np.linalg.norm(Y_-w_sk.T.dot(X_),ord='fro')/X_.shape[1],'\n\n')
```

```
my result
time= 0.10849595069885254  seconds
w=
 [[-5.06370205e-05  3.83915746e-04]
 [ 2.39266696e-01  6.62357767e-03]
 [-1.26168605e-03 -7.18129903e-05]
 [-1.44871660e-03  5.09803289e-04]]
MSE 8.221321832090663e-05


sklearn:
time= 0.12466740608215332  seconds
w=
 [[ 1.33213175e-06  3.77957792e-04]
 [ 2.39283423e-01  6.62165997e-03]
 [-1.26214423e-03 -7.17604620e-05]
 [-1.44939987e-03  5.09881622e-04]]
MSE 8.267688332411155e-05
```

3 Part b)

In this part we should note that the feature dimension is larger than number of samples. To make loading the files easier we have saved the tables of data and target in two separate files "data_x.csv" and "data_y.csv". You can skip the following chunk and run the next onw which computes the optimization.

[ ]:
```
## Load data and preprocessing

## Example of reading dataset ''crimedata.csv'' :
X_=np.ndarray((0,15*327))
Y_=np.ndarray((0,327))

#2921
for i in range(2920):
    print(i)
    f_name='ghg_data/ghg.gid.site'+format(i+2,'04d')+'.dat'
    #print(f_name)
    data = pd.read_csv(f_name, sep=" ",header=None)
    #print(data.shape)
    X = data.iloc[0:15,:]
    Y = data.iloc[15,:]

    X = np.array(X).reshape(1,X.size)
```

```
    Y = np.array(Y).reshape(1,Y.size)
    X[np.isnan(X)] = 0
    Y[np.isnan(Y)] = 0

    X = X.astype(float)
    Y = Y.astype(float)

    X_=np.append(X_,X,axis=0)
    Y_=np.append(Y_,Y,axis=0)

X_=X_.T
Y_=Y_.T

print(X_.shape)
print(Y_.shape)

pd.DataFrame(X_).to_csv("data_x.csv",index=None)
pd.DataFrame(Y_).to_csv("data_y.csv",index=None)
```

[36]:
```
X_=X_.T
Y_=Y_.T

print(X_.shape)
print(Y_.shape)

pd.DataFrame(X_).to_csv("data_x.csv",index=None)
pd.DataFrame(Y_).to_csv("data_y.csv",index=None)
```

```
(4905, 2920)
(327, 2920)
```

[42]:
```
#Load the data
#X_ = pd.read_csv('data_x.csv')
#Y_ = pd.read_csv('data_y.csv')

## Closed form solution and optimal linear regressor

# Define lambda here:
lam = 0.01# change the value

start = time.time()
## Calculate the closed-form solution here:
A=X_.T.dot(X_)+lam*np.eye(X_.shape[1])
w=X_.dot(np.linalg.inv(A)).dot(Y_.T)
end = time.time()
print('my result')
print('time=',end-start,' seconds')
print('w=\n',w)
```

```
print('MSE',np.linalg.norm(Y_-w.T.dot(X_),ord='fro')/X_.shape[1],'\n\n')


reg = linear_model.Ridge(alpha=0.01)
start = time.time()
## Find the optimal linear regressor here:
reg.fit(X_.T,Y_.T)
end = time.time()
w_sk=reg.coef_.T
print('sklearn:')
print('time=',end-start,' seconds')
print('w=\n',w_sk)
print('MSE',np.linalg.norm(Y_-w_sk.T.dot(X_),ord='fro')/X_.shape[1],'\n\n')
```

```
my result
time= 4.206740617752075  seconds
w=
 [[ 1.33803412 -4.31297239  4.16732297 ... -1.86285653 -5.94045856
  -2.42493391]
 [-0.6360965  -1.87888913 -2.78599494 ...  0.44103952  0.37084066
   0.79503398]
 [-1.68106993 -0.93070618  0.376551   ...  2.24363459  2.0820761
   3.31640699]
 ...
 [-0.19863554  2.77347287 -0.65526122 ...  3.29014362 -0.74947238
   2.05378872]
 [ 0.89627061 -0.47173865  0.5663919  ...  2.6731825  -1.16484659
   2.65890758]
 [ 0.40965161 -0.47173038  1.87818188 ... -4.50098456  6.77277378
  -6.85810515]]
MSE 0.0215563869078414


sklearn:
time= 1.6495862007141113  seconds
w=
 [[ 1.33300239 -4.30578355  4.16568257 ... -1.8672794  -5.93134384
  -2.41699773]
 [-0.59923742 -1.93154987 -2.77397844 ...  0.4734386   0.30407218
   0.73689869]
 [-1.51311191 -1.17066862  0.43130729 ...  2.39126951  1.77782807
   3.05149839]
 ...
 [-0.29996717  2.91824585 -0.68829652 ...  3.20107323 -0.56591493
   2.2136121 ]
 [ 0.8845912  -0.45505225  0.56258428 ...  2.66291633 -1.14368993
   2.67732869]
```

```
 [ 0.37120103 -0.41679586  1.86564654 ... -4.53478256  6.84242518
  -6.7974597 ]]
MSE 30.292344524156473
```

It can be observed that the time for running this part is more than the case with lower dimension. The reason of difference between MSE is that the norm of target itself is affecting the MSE and since the dimension of the target is high (327) the difference has become significant. Also two algorithms may compute the inverse differently.

4 part c)

If the dimension of the matrix is very high, we have to note two things: - The closed form solution should be handled if we do not have enough samples and X is tall. - To compute the inverse of a large matrix we can use techniques to take the inverse piecewise:

$$\mathbf{A} = \begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix}$$

Sounds great

and

$$\mathbf{A}^{-1} = \begin{pmatrix} \mathbf{E}^{-1} + \mathbf{E}^{-1}\mathbf{F}\mathbf{S}^{-1}\mathbf{G}\mathbf{E}^{-1} & -\mathbf{E}^{-1}\mathbf{F}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{G}\mathbf{E}^{-1} & \mathbf{S}^{-1} \end{pmatrix}.$$