EECS 545 Homework 4

Yuan Yin

November 1, 2018

Problem 1

(c)

The result is:

```
MSE for training data is: 19.016860651808518
MSE for test data is: 23.535860253927392
offset b is: 19.429475366627464
```

(d)

The result is:

```
MSE for training data is: 19.249564400341065
MSE for test data is: 26.396905605851654
```

Finding that for kernel ridge regression without offset, the errors both for training and test data are a little larger than kernel ridge regression with offset.

(e)

The code for (c) is:

```
import numpy as np
import scipy.io as sio

# Load the data and preprocessed the data
bodyfat_data = sio.loadmat('bodyfat_data.mat')
x = bodyfat_data['X']
y = bodyfat_data['y']
n,d = x.shape
train_num = 150; m = n - train_num
x_train = x[:train_num,:]; x_test = x[train_num:,:]
```

```
y_train = y[:train_num]; y_test = y[train_num:]
y_train_bar = np.mean(y_train)

# Helper function
def dist2(x,c):
    ndata,dimx = x.shape
    ncenters, dimc = c.shape

    xsum = np.sum(x**2,axis = 1)
    xsum = xsum[:,np.newaxis]
    csum = np.sum(c**2,axis = 1)
    csum = csum[:,np.newaxis]

    n2 =  xsum.dot(np.ones([1,ncenters]))+ np.ones([ndata,1]).dot(csum.T)- 2*x.dot(c.T)
    return n2

sigma = 15; lamda = 0.003
kernel = np.exp(-1/2/sigma**2 * dist2(x_train, x_train))
kernel_prime = np.exp(-1/2/sigma**2 * dist2(x_train, x_test))
O1 = 1/train_num * np.ones([train_num, m]); O2 = 1/train_num * np.ones([train_num,
    train_num])
kernel_tilda = kernel - kernel.dot(O2) - O2.dot(kernel) + (O2.dot(kernel)).dot(O2)
kernel_prime_tilda = kernel_prime - kernel.dot(O1) - O2.dot(kernel_prime) +
    (O2.dot(kernel)).dot(O1)

# Kernel ridge regression with offset
y_pre_train = []
for i in range(train_num):
    y_pre_train.append(y_train_bar + (np.mat(y_train - y_train_bar).T
                                      * np.mat(kernel_tilda + train_num * lamda *
                                          np.eye(train_num)).I
                                      * np.mat(kernel_tilda[:,i]).T)[0,0])
y_pre_test = []
for i in range(m):
    y_pre_test.append(y_train_bar + (np.mat(y_train - y_train_bar).T
                                      * np.mat(kernel_tilda + train_num * lamda *
                                          np.eye(train_num)).I
                                      * np.mat(kernel_prime_tilda[:,i]).T)[0,0])

# Compute MSE
MSE_train = sum([(y_train[i] - y_pre_train[i])**2 for i in range(train_num)])/train_num
MSE_test = sum([(y_test[i] - y_pre_test[i])**2 for i in range(n-train_num)])/(n-train_num)
print("MSE for training data is: ", MSE_train[0])
print("MSE for test data is: ", MSE_test[0])
```

```
# Compute offset b
b = y_train_bar - (np.mat(y_train - y_train_bar).T
                  * np.mat(kernel_tilda + train_num * lamda * np.eye(train_num)).I
                  * np.mat(kernel.dot(O2)[:,0] - ((O2.dot(kernel)).dot(O2))[0,0]).T)[0,0]
print("offset b is: ", b)
```

And the code for (d) is:

```
import numpy as np
import scipy.io as sio

# Load the data and preprocessed the data
bodyfat_data = sio.loadmat('bodyfat_data.mat')
x = bodyfat_data['X']
y = bodyfat_data['y']
n,d = x.shape
train_num = 150; m = n - train_num
x_train = x[:train_num,:]; x_test = x[train_num:,:]
y_train = y[:train_num]; y_test = y[train_num:]

# Helper function
def dist2(x,c):
    ndata,dimx = x.shape
    ncenters, dimc = c.shape

    xsum = np.sum(x**2,axis = 1)
    xsum = xsum[:,np.newaxis]
    csum = np.sum(c**2,axis = 1)
    csum = csum[:,np.newaxis]

    n2 =  xsum.dot(np.ones([1,ncenters]))+ np.ones([ndata,1]).dot(csum.T)- 2*x.dot(c.T)
    return n2

sigma = 15; lamda = 0.003
kernel = np.exp(-1/2/sigma**2 * dist2(x_train, x_train))
kernel_prime = np.exp(-1/2/sigma**2 * dist2(x_train, x_test))

# Kernel ridge regression without offset
y_pre_train = []
for i in range(train_num):
    y_pre_train.append(np.array(np.mat(kernel + train_num * lamda * np.eye(train_num)).I *
        np.mat(y_train)).T.dot(kernel[i,:].T))
```

```
y_pre_test = []
for i in range(m):
    y_pre_test.append(np.array(np.mat(kernel + train_num * lamda * np.eye(train_num)).I *
        np.mat(y_train)).T.dot(kernel_prime[:,i]))

# Compute MSE
MSE_train = sum([(y_train[i] - y_pre_train[i])**2 for i in range(train_num)])/train_num
MSE_test = sum([(y_test[i] - y_pre_test[i])**2 for i in range(n-train_num)])/(n-train_num)
print("MSE for training data is: ", MSE_train[0])
print("MSE for test data is: ", MSE_test[0])
```