Assignment4_Ridge_Regression

March 9, 2019

```
In [1]: # ECE 4950 Assignment 4

        ## Problem 4 (Linear Regression and Regularization)
```

0.0.1 You can either write your own code or use python package SKLearn in this assignment, other packages for linear regression are not allowed

In this assignment, we use the Concrete Compressive Strength Dataset (https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength) from UCI machine learning repository. We will explore how to use linear regression to predict the concrete compressive strength given the proportion of different materials you use to make high-performance concrete. Typical instances of this dataset looks like the following:

|Cement kg/m^3| Blast Furnace Slag kg/m^3 |Fly Ash kg/m^3| Water kg/m^3|Superplasticizer kg/m^3|Coarse Aggregate kg/m^3| Fine Aggregate kg/m^3|Age day| Concrete compressive strength MPa| |:------ :|:------:|:------:|:------:|:------:|:------:|:------:|:------:|:------:| | 540.0 | 0.0 |0.0 |162.0 |2.5 |1040.0| 676.0|28 |79.99 | |540.0 |0.0 | 0.0 | 162.0 | 2.5 |1055.0 | 676.0 | 28 |61.89| |266.0 |114.0 | 0.0 | 228.0 | 0.0 | 932.0 | 670.0 |28 | 45.85|

0.0.2 Data Splitting

We randomly split the dataset into two parts, the training set and the test set. The training set consists of 430 data points and the test set consists of 600 data points. Generally, the training dataset is larger than the test dataset. However, in this assignment, we make the training set to be smaller in order to highlight the influence of regularization. (Think about why regularization is more important when you have a smaller training set.) Using the following lines, we load all the training data into a numpy array *train* and all the testing data into a numpy array *test*.

```
In [4]: import numpy as np
        import pandas
        Trainfile = pandas.read_excel('Train.xlsx')
        Testfile = pandas.read_excel('Test.xlsx')
        train = Trainfile.values
        test = Testfile.values
        #Trainfile #output all the training samples in a table
```

The training inputs are stored in array X_{trn}, where $X_{trn}[i][j]$ represents the jth attribute of the ith instance in the training set. The training outputs are stored in array y_{trn}, where $y_{trn}[i]$ is the output of the ith instance in the training test. Similarly we store the inputs and outputs of the test set in X_{tst} and y_{tst}

```
In [5]: X_trn_raw, y_trn = train[:,:-1], train[:,-1:]
        X_tst_raw, y_tst = test[:,:-1], test[:,-1:]
```

0.0.3 Data Preprocessing

In order to better use linear regression, we shift normalize all the attributes of training input and testing input to make each attribute zero mean and norm no larger than 1. We are not normailizing the output. (Think about why.) After that, we append a bias attribute 1 to each of the input feartures.

```
In [6]: inputs = np.concatenate((X_trn_raw, X_tst_raw),axis = 0)
        inputs = inputs - np.mean(inputs,axis=0) #shift
        inputs = inputs/(np.max(inputs,axis=0)) #normalize
        inputs = np.concatenate((inputs, np.ones((X_trn_raw.shape[0]+X_tst_raw.shape[0],1))), a
        X_trn = inputs[:X_trn_raw.shape[0],:]
        X_tst = inputs[X_trn_raw.shape[0]:,:]
```

0.0.4 Let's get started

We do recommend you to understand the aforementioned steps since data preprocessing is a very important skill for a machine learning engineer. However, if you don't understand the previous pre-processing steps, it's totally fine. Now let's see what we get: 1. Training set $[X_{trn}, Y_{trn}]$, where all the input attributes are zero-mean and normalized. At the end of all the training inputs, there is a bias term 1 at the end. 2. Testing set $[X_{tst}, Y_{tst}]$ which has the same property as the training set.

0.0.5 (1) Linear regression (without regularization)

Let's look at the first task. You can either use functions in Sklearn or write your own code to train a linear regressor on the training set. The goal here is to find the w which minimizes the training loss

$$w^* = \arg\min_w \frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w \cdot x_i - y_i)^2 \tag{1}$$

Output the squre root of your average training loss and average test loss

$$l_{trn} = \sqrt{\frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w^* \cdot x_i - y_i)^2}, l_{tst} = \sqrt{\frac{1}{|S_{tst}|} \sum_{i \in S_{tst}} (w^* \cdot x_i - y_i)^2} \tag{2}$$

```
In [27]: # PART 1: Simple Linear Regression
         from sklearn.linear_model import LinearRegression
         # store your training loss in l_trn and testing loss in l_tst
         #edit within this range
         #FILL IN CODE HERE
         reg = LinearRegression().fit(X_trn, y_trn)
         y_trn_predict = reg.predict(X_trn)
         #testing
         y_tst_predict = reg.predict(X_tst)
         #
```

```
        l_trn = np.sqrt(np.sum(np.square(y_trn_predict-y_trn))/y_trn.size)
        l_tst = np.sqrt(np.sum(np.square(y_tst_predict-y_tst))/y_tst.size)
        #end edit
        print('The average training loss is: ', l_trn)
        print('The average test loss is: ', l_tst)
```

```
The average training loss is:  9.727779122009025
The average test loss is:  10.904551016681253
```

0.0.6 (2) Linear Ridge regression

You may observe that training error is smaller than the test error. This is because we are overfitting to the training data. Regularization is a good way to avoid overfitting. Recall that λ- Ridge regression is to add an additional ℓ_2-norm of the weight vector to refine the obejective function:

$$w^{\lambda} = \arg\min_{w} \frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w \cdot x_i - y_i)^2 + \lambda |w|^2 \tag{3}$$

Try Ridge regression on the same dataset (the dataset with more attributes) under $\lambda = 0.001 \times (1.2)^i, i = 0, 1, 2, 3, ..., 99$. Plot the training loss and test loss under different regression parameter and answer the flowing questions: 1. Describe the trend of training error? (Increase w.r.t λ, decrese w.r.t λ, remain constant, first increase then decrease or first decrease then increase). Try to explain.

2. Describe the trend of testing error? (Increase w.r.t λ, decrese w.r.t λ, remain constant, first increase then decrease or first decrease then increase). Try to explain

Training and testing error are defined without the regularization term:

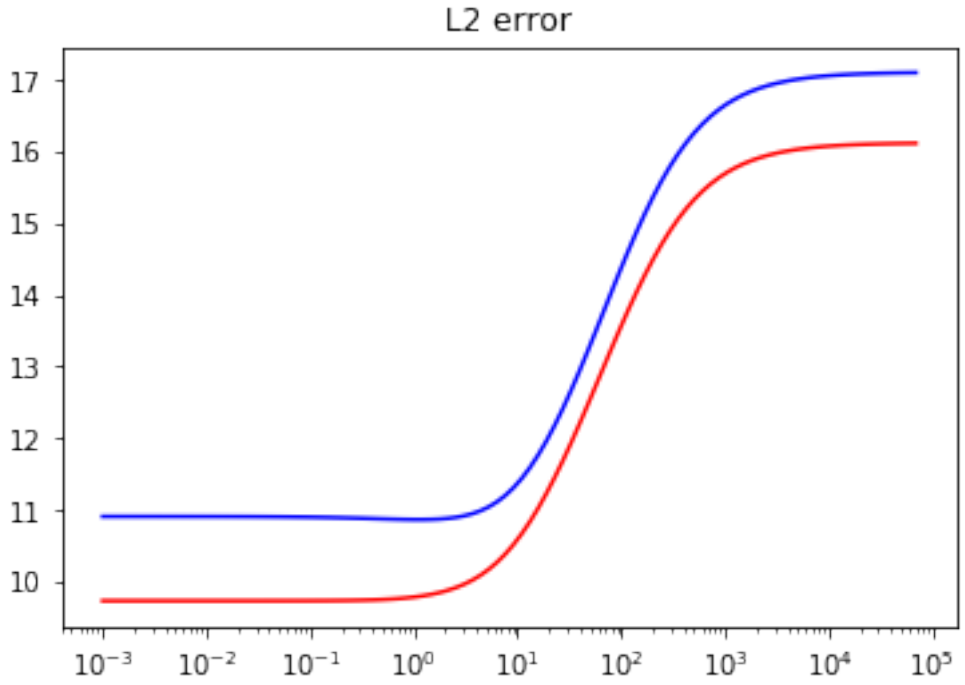
$$l^{\lambda}_{trn} = \sqrt{\frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w^{\lambda} \cdot x_i - y_i)^2}, l^{\lambda}_{tst} = \sqrt{\frac{1}{|S_{tst}|} \sum_{i \in S_{tst}} (w^{\lambda} \cdot x_i - y_i)^2} \tag{4}$$

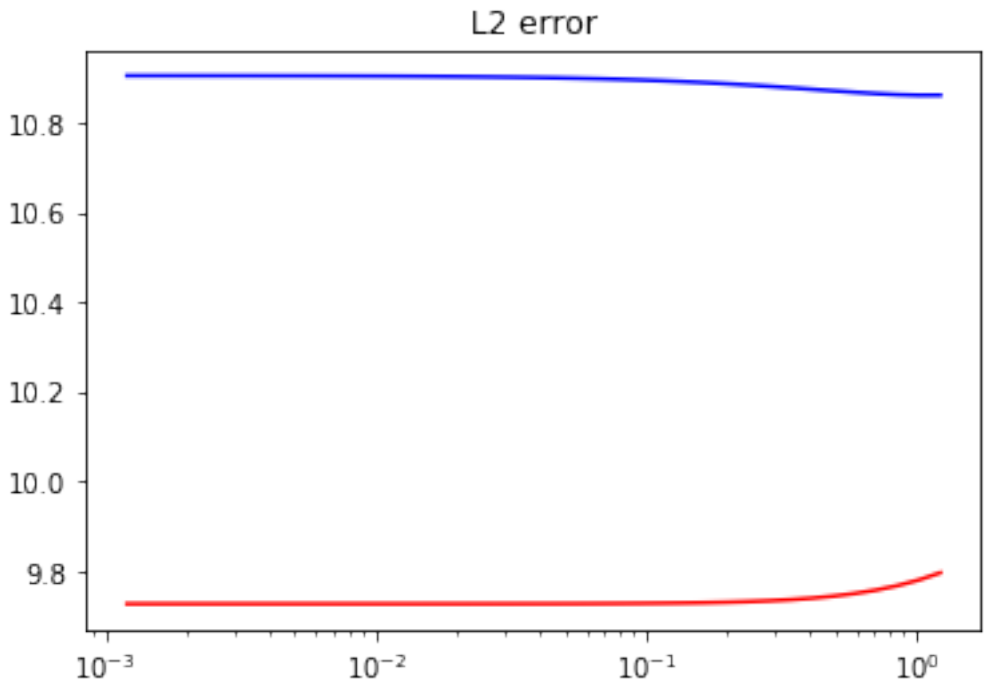
```
In [64]: # PART 2: Linear Ridge Regression
         from sklearn.linear_model import Ridge
         # In this part, you will perform linear Ridge Regression.
         # You will plot the training and test error as a function of lambda
         import matplotlib.pyplot as plt
         %matplotlib inline
         lamb = []
         points = 100
         for i in range(points):
             lamb.append(0.001*(1.2**i))
             #= 0.001 * [1, 1.2, 1.44, ..., 1.2^99] #lambda values #lambda values
         err_trn = [] # store your training error in this vector
         err_tst = [] # store your testing error in this vector
         #edit within this range
         #FILL IN CODE HERE
         for i in range(points):
             lamb.append(0.001*(1.2**i))
```

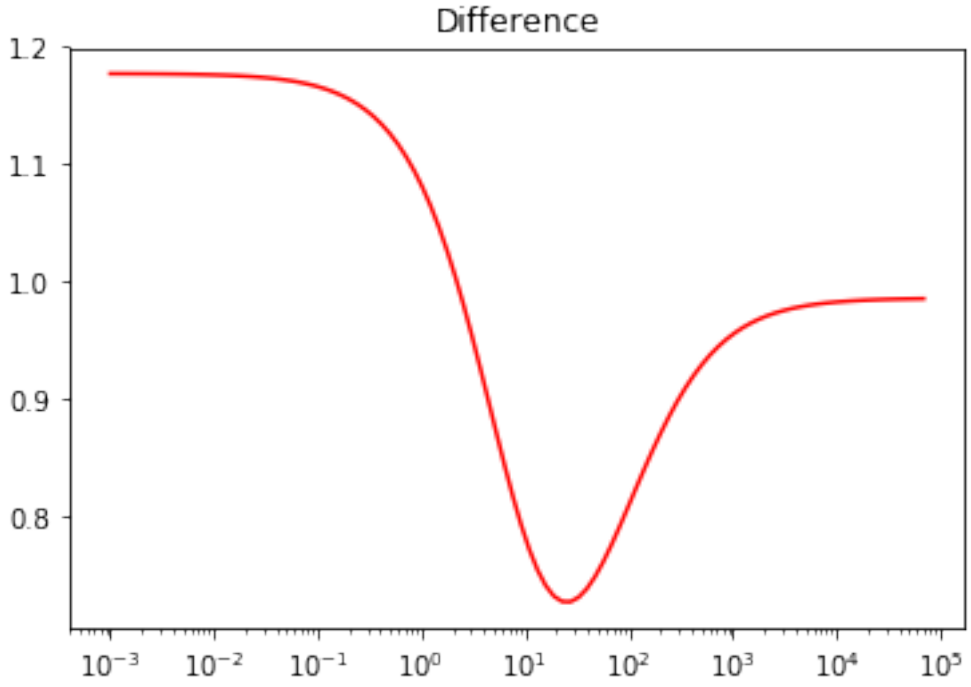
```
        ridge = Ridge(alpha=lamb[i]).fit(X_trn, y_trn)
        y_trn_predict = ridge.predict(X_trn)
        err_trn.append(i)
        err_trn[i] = np.sqrt(np.sum(np.square(y_trn_predict-y_trn))/y_trn.size)
        y_tst_predict = ridge.predict(X_tst)
        err_tst.append(i)
        err_tst[i] = np.sqrt(np.sum(np.square(y_tst_predict-y_tst))/y_tst.size)
    #end edit
```

In [72]:
```
diff = np.subtract(err_tst,err_trn)
```

In [73]:
```
plt.semilogx(lamb[0:100],err_trn,color='red')
plt.semilogx(lamb[0:100],err_tst,color='blue')
plt.title("L2 error")
plt.show()
plt.semilogx(lamb[1:40],err_trn[1:40],color='red')
plt.semilogx(lamb[1:40],err_tst[1:40],color='blue')
plt.title("L2 error")
plt.show()
plt.semilogx(lamb[0:100],diff,color='red')
plt.title("Difference")
plt.show()
```







Both trainng and testing erros does not change too much then grow and remain constant; However I plotted the difference in these 2 errors, to get a sense of generaization of lama, you can

see it drops first then increase.

0.0.7 Add more attributes

Can we do more beyond simple linear regression? Since the relationship between each attribute and the final output may not necessarily be linear, we may do better if we add some non-linearity. We can add the square of each attribute to our training input and do linear regression on the new set of attributes.

Remark: This is called polynomial regression, please google it if you are interested.

```
In [74]: inputs = np.concatenate((X_trn_raw, X_tst_raw),axis = 0)
         inputs_new = np.concatenate((inputs,np.square(inputs)),axis=1) #add square term
         inputs_new = inputs_new - np.mean(inputs_new,axis=0) #shift
         inputs_new = inputs_new/(np.max(inputs_new,axis=0)) #normalize
         inputs_new = np.concatenate((inputs_new, np.ones((X_trn_raw.shape[0]+X_tst_raw.shape[
         X_trn_new = inputs_new[:X_trn_raw.shape[0],:]
         X_tst_new = inputs_new[X_trn_raw.shape[0]:,:]
```

0.0.8 (3) Quadratic regression (without regularization)

Recall now we have X_{trn}^{new} as our new training features and X_{tst}^{new} as our new test features.

$$x_i^{new} = (x_i[1], x_i[2], ..., x_i[d], x_i[1]^2, x_i[2]^2, ..., x_i[d]^2) \tag{5}$$

Do the same thing as in Problem (1). Find the minimizer of the training error:

$$w^* = \arg\min_w \frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w \cdot x_i^{new} - y_i)^2 \tag{6}$$

Output the squre root of your average training loss and average test loss

$$l_{trn} = \sqrt{\frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w^* \cdot x_i^{new} - y_i)^2}, l_{tst} = \sqrt{\frac{1}{|S_{tst}|} \sum_{i \in S_{tst}} (w^* \cdot x_i^{new} - y_i)^2} \tag{7}$$

```
In [76]: #PART 3: Quatratic Regression
         # store your training loss in l_trn and testing loss in l_tst
         #edit within this range
         # store your training loss in l_trn and testing loss in l_tst
         #edit within this range
         #FILL IN CODE HERE
         poly = LinearRegression().fit(X_trn_new, y_trn)
         y_trn_predict = poly.predict(X_trn_new)
         #testing
         y_tst_predict = poly.predict(X_tst_new)
         #
         l_trn = np.sqrt(np.sum(np.square(y_trn_predict-y_trn))/y_trn.size)
         l_tst = np.sqrt(np.sum(np.square(y_tst_predict-y_tst))/y_tst.size)
         #end edit
         print('The average training loss is: ', l_trn)
         print('The average test loss is: ', l_tst)
```

```
The average training loss is:  7.481820649524704
The average test loss is:  8.871400924312628
```

0.0.9 (4) Quadratic Ridge regression

In this problem, we do the same regularization as in Problem (2) to our new dataset:

Recall that λ- Ridge regression is to add an additional ℓ_2-norm of the weight vector to refine the obejective function:

$$w^{\lambda} = \arg\min_{w} \frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w \cdot x_i^{new} - y_i)^2 + \lambda |w|^2 \tag{8}$$

Try Ridge regression on the new dataset (the dataset with more attributes) under $\lambda = 0.001 \times (1.2)^i, i = 0, 1, 2, 3, ..., 99$. Plot the training loss and test loss under different regression parameter and answer the flowing questions: 1. Describe the trend of training error? (Increase w.r.t λ, decrese w.r.t λ, remain constant, first increase then decrease or first decrease then increase). Try to explain.

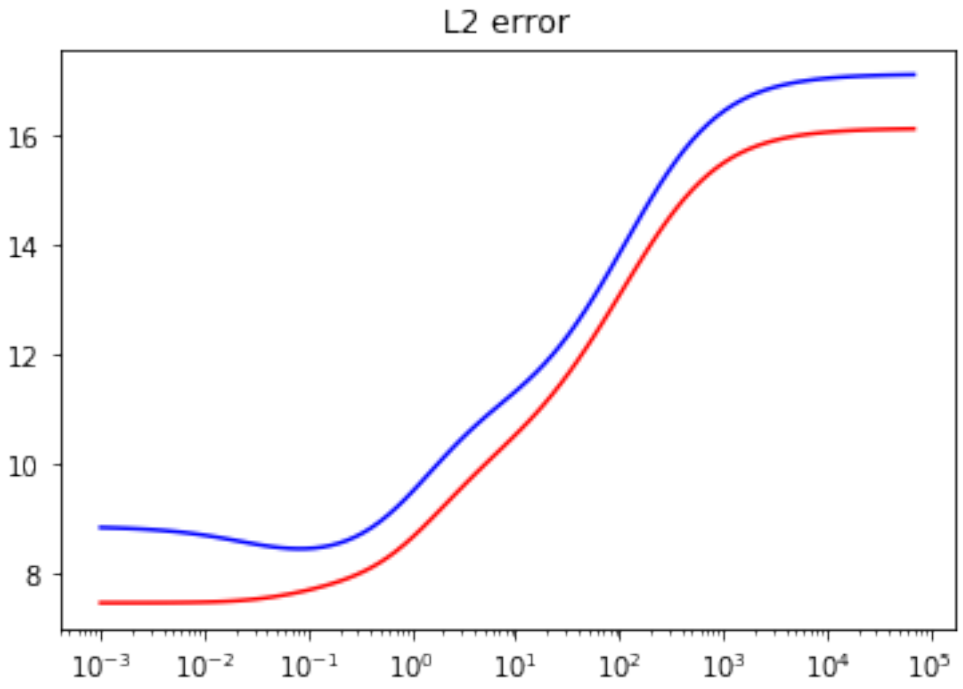
2. Describe the trend of testing error? (Increase w.r.t λ, decrese w.r.t λ, remain constant, first increase then decrease or first decrease then increase). Try to explain

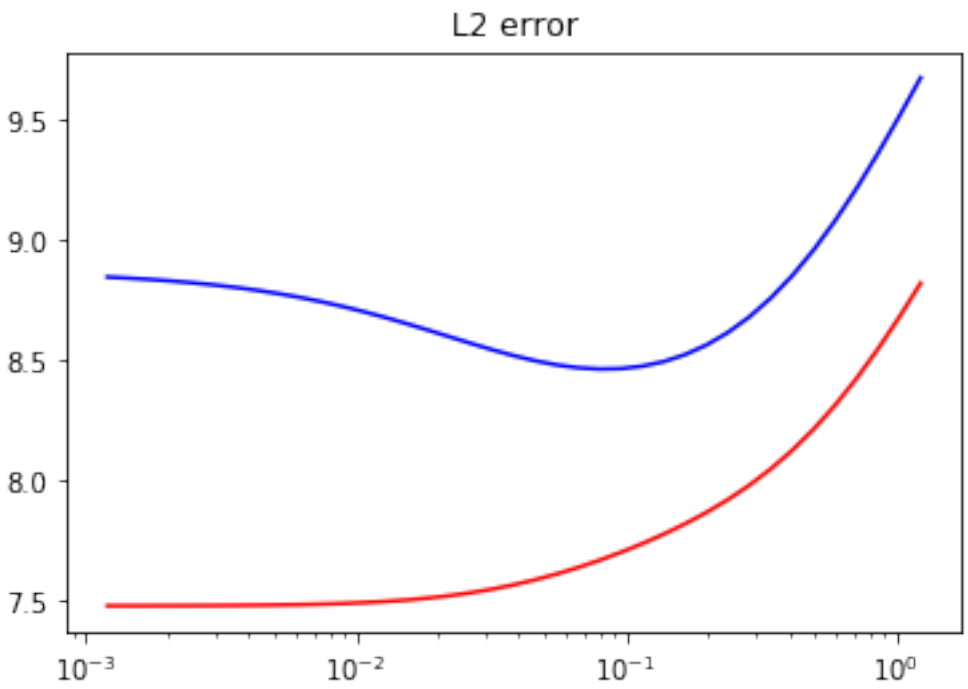
Training and testing error are defined without the regularization term:

$$l_{trn}^{\lambda} = \sqrt{\frac{1}{|S_{trn}|} \sum_{i \in S_{trn}} (w^{\lambda} \cdot x_i^{new} - y_i)^2}, l_{tst}^{\lambda} = \sqrt{\frac{1}{|S_{tst}|} \sum_{i \in S_{tst}} (w^{\lambda} \cdot x_i^{new} - y_i)^2} \tag{9}$$

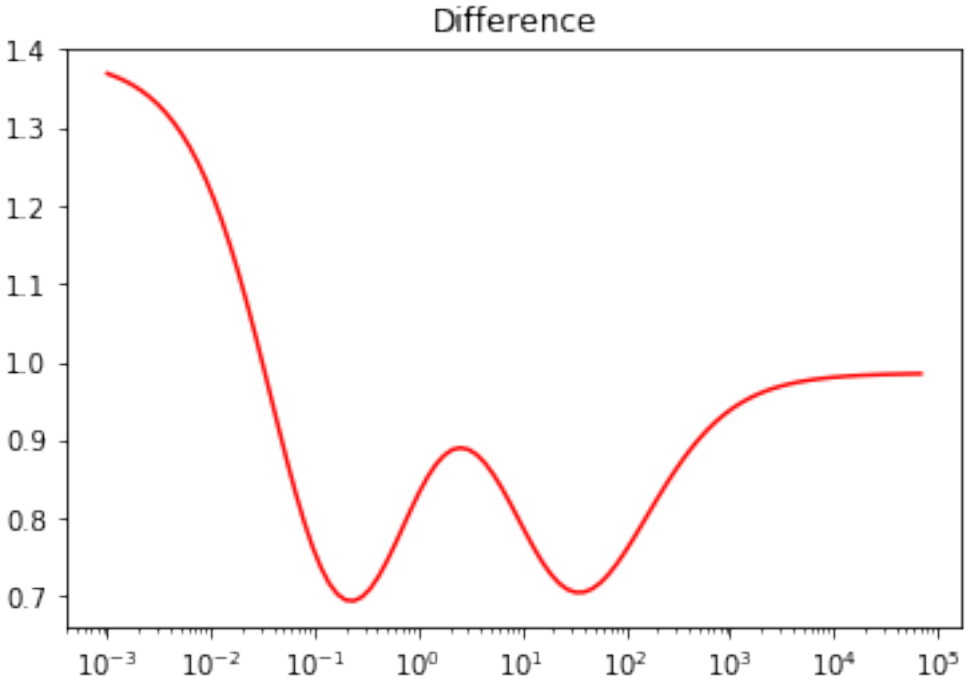
In [79]:
```
# PART 4: Quadratic Ridge Regression

# In this part, you will perform Ridge Regression.
# You will plot the training and test error as a function of lambda

import matplotlib.pyplot as plt
%matplotlib inline
lamb = []
points = 100
for i in range(points):
    lamb.append(0.001*(1.2**i))
    #= 0.001 * [1, 1.2, 1.44, ..., 1.2^99] #lambda values
err_trn = [] # store your training error in this vector
err_tst = [] # store your testing error in this vector
#edit within this range
for i in range(points):
    lamb.append(0.001*(1.2**i))
    quadra = Ridge(alpha=lamb[i]).fit(X_trn_new, y_trn)
    y_trn_predict = quadra.predict(X_trn_new)
    err_trn.append(i)
    err_trn[i] = np.sqrt(np.sum(np.square(y_trn_predict-y_trn))/y_trn.size)
    y_tst_predict = quadra.predict(X_tst_new)
    err_tst.append(i)
```

```
    err_tst[i] = np.sqrt(np.sum(np.square(y_tst_predict-y_tst))/y_tst.size)
#end edit
plt.semilogx(lamb[0:100],err_trn,color='red')
plt.semilogx(lamb[0:100],err_tst,color='blue')
plt.title("L2 error")
plt.show()
plt.semilogx(lamb[1:40],err_trn[1:40],color='red')
plt.semilogx(lamb[1:40],err_tst[1:40],color='blue')
plt.title("L2 error")
plt.show()
```






```
In [80]: diff = np.subtract(err_tst,err_trn)
         plt.semilogx(lamb[0:100],diff,color='red')
         plt.title("Difference")
         plt.show()
```



0.0.10 (5) Discussion

Compare the plots of Linear Regression and Quadratic Regression

So it is very interesting that Quadratic Regression we see more variantion in the errors: 1. Training error decrease then increase, testing error keeps growing (remain relatively constant in the beginning) 2. for linear regression the performace is relatively singular, quadratic or poly is not. 3. error of Quaratic regression/poly is smaller, but if keep growing dimension will encounter curse of dimension