EE769 Introduction to Machine Learning

Assignment 1: Gradient Descent, Linear Regression, and Regularization

Template and Instructions

1. Up to two people can team up, but only one should submit, and both should understand the entire code.
2. Every line of code should end in a comment explaining the line
3. It is recommended to solve the assignment in Google Colab. Write your roll no.s separated by commas here:
4. Write your names here:
5. There are two parts to the assignment. In the Part 1, the code format has to be strictly followed to enable auto-grading. In the second part, you can be creative.
6. **You can discuss with other groups or refer to the internet without being penalized, but you cannot copy their code and modify it. Write every line of code and comment on your own.**

Part 1 begins ...

Instructions to be strictly followed:

1. Do not add any code cells or markdown cells until the end of this part. Especially, do not change the blocks that say "TEST CASES, DO NOT CHANGE"
2. In all other cells only add code where it says "CODE HERE".
3. If you encounter any raise NotImplementedError() calls you may comment them out.

We cannot ensure correct grading if you change anything else, and you may be penalised for not following these instructions.

Import Statements

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

Normalize function

Add your code in the cell below to normalize the independent variables, making them zero mean and unit variance.

```
def Normalize(X): # Output should be a normalized data matrix of the same dimension
    '''
    Normalize all columns of X using mean and standard deviation
    '''
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 - 1 dimensional array'''
#X=np.array([[1,2,3],[3,4,5],[7,8,9]])
X1=np.array([1,2,3])
np.testing.assert_array_almost_equal(Normalize(X1),np.array([-1.224,  0.      ,  1.224]),d
''' case 2 - 2 dimensional array'''
X2=np.array([[4,7,6],[3,8,9],[5,11,10]])
np.testing.assert_array_almost_equal(Normalize(X2),np.array([[ 0.  , -0.980581, -1.372813]
''' case 3 - 1 dimensional array with float'''
X3=np.array([5.5,6.7,3.2,6.7])
np.testing.assert_array_almost_equal(Normalize(X3),np.array([-0.017,  0.822, -1.627,  0.82
```

Prediction Function

Given X and w, compute the predicted output. Do not forget to add 1's in X

```
def Prediction (X, w): # Output should be a prediction vector y
    '''
    Compute Prediction given an input datamatrix X and weight vecor w. Output y = [X 1]w w
    '''
    # YOUR CODE HERE
    raise NotImplementedError()

```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 - Known input output matrix and weights 1'''
X1 = np.array([[3,2],[1,1]])
w1 = np.array([2,1,1])
```

```
np.testing.assert_array_equal(Prediction(X1,w1),np.array([9,4]))

```

Loss Functions

Code the four loss functions:

1. MSE loss is only for the error
2. MAE loss is only for the error
3. L2 loss is for MSE and L2 regularization, and can call MSE loss
4. L1 loss is for MSE and L1 regularization, and can call MSE loss

```
def MSE_Loss (X, t, w, lamda =0): # Ouput should be a single number
    '''
    lamda=0 is a default argument to prevent errors if you pass lamda to a function that d
    This allows us to call all loss functions with the same input format.

    You are encouraged read about default arguments by yourself online if you're not famil
    '''
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_almost_equal(MSE_Loss(X,t,w),0.53,decimal=3)

```

```
def MAE_Loss (X, t, w, lamda = 0): # Output should be a single number
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_almost_equal(MAE_Loss(X,t,w),0.700,decimal=3)

```

```
def L2_Loss (X, t, w, lamda=0): # Output should be a single number
    ''' Need to specify what inputs are'''
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_almost_equal(L2_Loss(X,t,w,0.5),1.675,decimal=3)

```

```
def L1_Loss (X, t, w, lamda): # Output should be a single number
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_almost_equal(L1_Loss(X,t,w,0.5),2.280,decimal=3)

```

```
def NRMSE_Metric (X, t, w): # Output should be a single number
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' Test case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_almost_equal(NRMSE_Loss(X,t,w),0.970,decimal=3)

```

Gradient function

Each Loss function will have its own gradient function:

1. MSE gradient is only for the error
2. MAE gradient is only for the error
3. L2 gradient is for MSE and L2 regularization, and can call MSE gradient
4. L1 gradient is for MSE and L1 regularization, and can call MSE gradient

```
def MSE_Gradient (X, t, w, lamda=0):
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_array_almost_equal(MSE_Gradient(X,t,w),np.array([2.55, 2.94, 2.9 , 0.4 ]
```

```
def MAE_Gradient (X, t, w, lamda=0): # Output should have the same size as w
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
np.testing.assert_array_almost_equal(MAE_Gradient(X,t,w),np.array([0.75,  0.3 ,  0.5 , 0.]
```

```
def L2_Gradient (X, t, w, lamda): # Output should have the same size as w
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 '''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
w=np.array([2,-1,0.5,1])
```

```
8 np.testing.assert_array_almost_equal(L2_Gradient(X,t,w,0.5),np.array([2.986, 2.721, 3.009
9
```

```
1 def L1_Gradient (X, t, w, lamda): # Output should have the same size as w
2     # YOUR CODE HERE
3     raise NotImplementedError()
```

```
1 '''
2 TEST CASES, DO NOT CHANGE
3 '''
4 ''' case 1 '''
5 X=np.array([[3,6,5],[4.5,6.6,6]])
6 t=np.array([4,5.5])
7 w=np.array([2,-1,0.5,1])
8 np.testing.assert_array_almost_equal(L1_Gradient(X,t,w,0.5),np.array([3.05, 2.44, 3.4 , 0.
9
```

Gradient Descent Function

```
1 def Gradient_Descent (X, X_val, t, t_val, w, lamda, max_iter, epsilon, lr, lossfunc, gradf
2     # YOUR CODE HERE
3     raise NotImplementedError()
4     return w_final, train_loss_final, validation_loss_final, validation_NRMSE #You should
```

```
1 '''
2 TEST CASES, DO NOT CHANGE
3 '''
4 np.random.seed(2)
5
6 X=np.array([[23,24],[1,2]])
7 t=np.array([4,5])
8 X_val=np.array([[3,4],[5,6]])
9 t_val=np.array([3,4])
10 w=np.array([3,2,1])
11 results =Gradient_Descent (X, X_val, t, t_val, w, 0.1, 100, 1e-10, 1e-5, L2_Loss,L2_Gradie
12 np.testing.assert_array_almost_equal([results[1],results[2]],[697.919,17.512],decimal=3)
```

Pseudo Inverse Method

You have to implement a slightly more advanced version, with L2 penalty:

w = (X' X + lambda I)^(-1) X' t.

See, for example: Section 2 of https://web.mit.edu/zoya/www/linearRegression.pdf

```
def Pseudo_Inverse (X, t, lamda): # Output should be weight vector
    # YOUR CODE HERE
    raise NotImplementedError()
```

```
'''
TEST CASES, DO NOT CHANGE
'''
''' case 1 - other data'''
X=np.array([[3,6,5],[4.5,6.6,6]])
t=np.array([4,5.5])
np.testing.assert_array_almost_equal(Pseudo_Inverse(X,t,0.5),np.array([ 0.491,  0.183,  0.
```

Save the code above this as a RollNo1_RollNo2_1.py file after running the test blocks to make sure there are no errors.

... Part 1 ends

Below this you be more creative. Just comment out the lines where you save files (e.g. test predictions).

Part 2 begins ...

Instructions to be loosely followed (except number 8):

1. Add more code and text cells between this and the last cell.
2. Read training data from: https://www.ee.iitb.ac.in/~asethi/Dump/TempTrain.csv only. Do not use a local copy of the dataset.
3. Find the best lamda for **MSE+lamda*L2(w)** loss function. Plot training and validation RMSE vs. 1/lamda (1/lamda represents model complexity). Print weights, validation RMSE, validation NMSE for the best lamda.
4. Find the best lamda for **MSE+lamda*L1(w)** loss function. Plot training and validation RMSE vs. 1/lamda (1/lamda represents model complexity). Print weights, validation RMSE, validation NMSE for the best lamda.
5. Find the best lamda for the **pseudo-inv method**. Plot training and validation RMSE vs. 1/lamda (1/lamda represents model complexity). Print weights, validation RMSE, validation NMSE for the best lamda.
6. Write your observations and conclusions.

7. Read test data from: https://www.ee.iitb.ac.in/~asethi/Dump/TempTest.csv only. Do not use a local copy of the dataset. Predict its dependent (missing last column) using the model with the lowest MSE, RMSE, or NMSE. Save it as a file RollNo1_RollNo2_1.csv.
8. **Disable the prediction csv file saving statement and submit this entire .ipynb file (part 1 and part 2), .py file (part 1 only), and .csv file as a single RollNo1_RollNo2_1.zip file.**

... Part 2 ends.

1. Write the name or roll no.s of friends from outside your group with whom you discussed the assignment here (no penalty for mere discussion without copying code):
2. Write the links of sources on the internet referred here (no penalty for mere consultation without copying code):