EE6310

Problem Set 9

We will utilize the data set from the Problem Set 7 and implement the regularized lomear regression.

Name	ΔH_f [eV]	# CH_3	# CH_2	# CH	# C
ethane	-0.87	2	0	0	0
propane	-1.08	2	1	0	0
…					
3,4-ditert-butyl-2,2,5,5-tetramethylhexane	-2.60	12	0	2	4

In python code, the following code can be copied and pasted to make a numpy array.

```
import numpy as np
data = np.array([[-0.87,2,0,0,0],[-1.08,2,1,0,0],[-1.30,2,2,0,0],[-
1.52,2,3,0,0],[-1.73,2,4,0,0],[-1.95,2,5,0,0],[-2.16,2,6,0,0],[-
2.37,2,7,0,0],[-2.59,2,8,0,0],[-2.80,2,9,0,0],[-3.01,2,10,0,0],[-
3.23,2,11,0,0],[-3.44,2,12,0,0],[-3.68,2,13,0,0],[-3.89,2,14,0,0],[-
4.08,2,15,0,0],[-4.30,2,16,0,0],[-4.51,2,17,0,0],[-4.72,2,18,0,0],[-
7.22,2,30,0,0],[-1.41,3,0,1,0],[-1.59,3,1,1,0],[-1.81,3,2,1,0],[-
1.78,3,2,1,0],[-2.02,3,3,1,0],[-1.99,3,3,1,0],[-1.97,3,3,1,0],[-
2.23,3,4,1,0],[-2.20,3,4,1,0],[-2.20,3,4,1,0],[-2.19,3,4,1,0],[-
2.70,3,6,1,0],[-2.68,3,6,1,0],[-6.15,3,22,1,0],[-6.09,3,22,1,0],[-
7.25,3,27,1,0],[-1.74,4,0,0,1],[-1.92,4,1,0,1],[-1.84,4,0,2,0],[-
2.14,4,2,0,1],[-2.06,4,1,2,0],[-2.09,4,1,2,0],[-2.09,4,2,0,1],[-
2.33,4,3,0,1],[-2.22,4,2,2,0],[-2.27,4,2,2,0],[-2.31,4,2,2,0],[-
2.28,4,3,0,1],[-2.21,4,2,2,0],[-2.19,4,2,2,0],[-2.23,4,3,0,1],[-
2.55,4,4,0,1],[-2.12,5,0,1,1],[-2.28,5,1,1,1],[-2.32,5,1,1,1],[-
2.24,5,1,1,1],[-2.25,5,0,3,0],[-2.62,5,2,1,1],[-2.51,5,1,3,0],[-
2.36,5,1,3,0],[-2.34,6,0,0,2],[-2.50,6,1,0,2],[-2.95,6,2,0,2],[-
2.75,6,6,0,2],[-2.45,9,0,1,3],[-2.57,10,0,0,4],[-2.60,12,0,2,4]])
```

(a) We will implement the regularization with the training, validation and test set. The first step is to split the data into training set, validation set, and test set. It is important randomize the data, so we are not introducing bias to the data splitting. numpy.random.shuffle shuffles the data along the first dimension. Use it to shuffle the data.

(b) Split the data into training, validation and test set, which we will call as "data_train", "data_validation", and "data_test". We will have to decide what ratio to split the data into. Let "data_train" be 80% of the data, "data_validation" to be 10% of the data, and "data_test" to be another 10% of the data.

(c) Split the data_train, data_validatoin, data_test into x and y. You will need x and y for the training set, validation set, and the test set each. Thus, let's set them as x_{train}, x_{val}, x_{test} and y_{train}, y_{val}, y_{test}. You can set them as x_train, x_val, x_test, y_train, y_val, and y_test in code.

(d) We will perform normalization based on the mean and standard deviation of the training set. Calculate the mean and standard deviation of the training set as x_mean and x_std, and use it to normalize validation and test set.

(e) Write the cost function and the gradient descent code. You need to modify the code from Problem set 8 slightly. It will need additional input of lambda.

```
def cost_function(w,b,x,y,lamb):
    # your code
    return J

print(cost_function(np.ones(79),1,x_train,y_train,1).sum()) # output should be 1.111441285379255
```

```
def gradient_descent(w,b,x,y,alpha,lamb):
    # your code
    return w,b
w,b = gradient_descent(np.ones(79),1,x_train,y_train,0.01,1)
print(w.sum(),b.sum()) # output should be  79.08383583452996 0.9975526486892478
```

(f) We are now going to write the algorithm for the model selection and the testing as well. Here we will just try to find the best λ. The introduce pseudo code is:

1. For various λ:
2. Minimize $J_{\mathbf{w},b}^{(\text{train})}$ with gradient descent to get $\mathbf{w}, b$
3. Calculate the validation set cost ($J_{\mathbf{w},b}^{(\text{val})}$)
4. Using the $f_{\mathbf{w},b}(x)$ and λ with the lowest $J_{\mathbf{w},b}^{(\text{val})}$, calculate the test set cost ($J_{\mathbf{w},b}^{(\text{test})}$)

Use the λ as shown below:

```
lambdas = 10**(np.arange(-20,0,0.5))
```

For a more detailed pseudocode is:

1. For λ in `10**(np.arange(-20,0,0.5))`:
2. Initialize $\mathbf{w}, b$
3. For i increasing from 1 to the maximum number of iterations: # copy paste from the problem set 7
4. $\mathbf{w}, b$ = gradient_descent
5. Stop gradient descent if the J_{train} decreased by less than 10^{-11}
6. Calculate J_{val} and record it into a list
7. Save the $\mathbf{w}, b$ for this λ
8. Find the λ with the minimum J_{val}
9. Get the $\mathbf{w}, b$ with the minimum J_{v} # use np.argmin
10. Calculate the J_{test} with the $\mathbf{w}, b$

What is the best λ? Note that the answer will change for each run as you are randomly shuffling the data.