CS165B: Intro. to Machine Learning, Fall 2020
Programming assignment # 1

Due: Sunday 11:59pm, October 25th

IMPORTANT NOTE You can discuss the problems with your TAs and instructor, but all answers and codes written down and turned in must be your own work.

We will use Kaggle data set "Graduate Admission 2" https://www.kaggle.com/mohansacharya/graduate-admissions for this assignment. The dataset contains a number of parameters that are considered important during the application for Master's programs in India. The parameters included are : 1. GRE Scores (out of 340), 2. TOEFL Scores (out of 120), 3. University Rating (out of 5), 4. Statement of Purpose (out of 5), 5. Letter of Recommendation Strength (out of 5), 6. Undergraduate GPA (out of 10), 7. Research Experience (either 0 or 1), and 8. Chance of Admit (ranging from 0 to 1). It can be expected that roughly linear relations exist among factors (1) to (7) and (8). Furthermore, often times, say, GRE scores and GPA show a high degree of correlation. Hence, it is reasonable to attempt a multi-variate linear regression with regularization for this problem. You can use Panda library to query dataset information.

The programming part of the assignment is then for you to implement an iterative stochastic gradient descent (SGD) linear regressor with regularization. You must use Python for this assignment (so the reader can grade your assignment automatically using Gradescope), and you must implement the iterative solver yourself (not calling an existing package). Your solver must support 2D grid search with one dimension being the learning rate α and the other dimension being the regularization weight λ. The skeleton class (SGDSolver) will be provided for you (lambda will be renamed lam as lambda is a keyword in Python).

```
class SGDSolver():

    def __init__(self, path):

    def training(self, alpha, lam, nepoch, epsilon):

    def testing(self, testX) :
```

Critical variables and parameters are explained below:

- *self.x*: a numpy array of size $n \times k$, with n being the number of training samples, and k being the number of features per training sample. In our case $k = 7$.
- *self.y*: a numpy array of size $n \times 1$ of the ground-truth regression values, with n being the number of training samples. *self.x* and *self.y* come from the *path* parameter in the constructor that is the name of the numpy file with the training data.
- *alpha*: range of the learning rates in the grid search (a list of two floats $[\alpha_{min}, \alpha_{max}]$)
- *lam*: range of the regularization weights in the grid search (a list of two floats $[\lambda_{min}, \lambda_{max}]$)
- *nepoch*: maximum number of training epochs (an integer). Each epoch comprising going through all training samples in a random order once. For SGD, the *batchsize* parameter (how many samples are used together in training) is set to 1, or training samples are used individually.

- *epsilon*: error bound to stop iteration prematurely if loss e_{mse} (see below) is less than the given bound (a float).

Note that in your implementation, in addition to weight w (one for each input features) you should also include bias (b) as your class variable. However, regularization applies only to w not to b. The loss (error) in regression is defined as the mean squared error (MSE) between the ground truth values and your regression values: Denote the ith component of the y vector as y^i, then $e_{mse} = \frac{1}{n}\sum_{i=1}^{n}(y^i_{GT} - y^i_{regress})^2$.

In this project, you will only receive 90% of the original dataset for training. 10% data will be held back for Gradescope use to evaluate your performance. Your training and validation codes must rely entirely on the given data (90% of the original data) to train your model. How you split the given data for training and validation, e.g., using n-fold validation, is up to you.

Your SGDSolver with be tested by three calls in succession: (1) initialization, (2) training, and (3) testing.

- During the initialization, you will be given the file name containing the training data. You are to populate self.x and self.y from the input file.
- During the training phase, your function *training* will be called with $alpha, lam, nepoch$ and *epsilon* given. If you need any other parameters for your code to work, it is entirely up to you to define these parameters and provide some reasonable default values for them. You can define as many auxiliary functions inside SGDSolver as you see fit. The training phase must start from scratch with a randomized initial guess to w and b. The function should store both weight and bias within the SGDSolver class after the training.

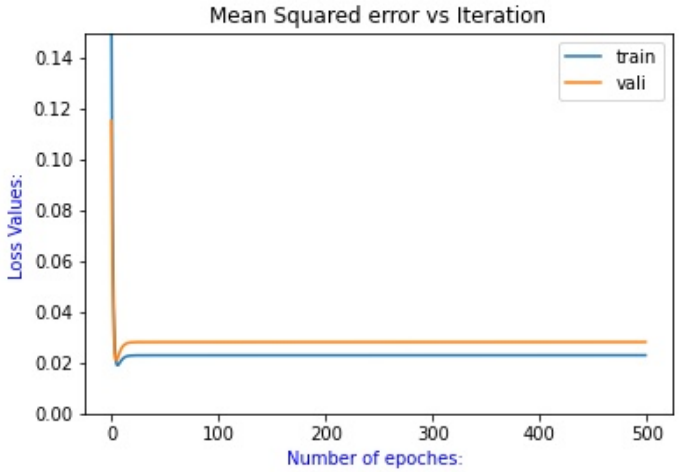
  For example,

  ```
  model = SGDSolver('Admission_Predict.csv')
  model.training([1e-06, 0.1], [0.01, 10.0], 100, 1e-05)
  ```

- During the testing phase, your function *testing* will be called with a testX parameter that is an $n \times k$ numpy array. You should return the regression result in the form of an $n \times 1$ numpy array testY. For example,

  ```
  testY = model.testing(testX)
  ```

It is highly recommended that you implement a graphing function that graphs the training and validation errors as functions of iterations as shown in the figure below. Such a graph is very useful in visualizing your program performance and is a good exercise for you to use the pyplot package if you have never used it before. Your program should finish its training phase in a "reasonable" amount of time on the given training data. By "reasonable," we mean less than half an hour on Gradescope. The codes you turn in must NOT save any parameters as the Gradescope will test your program from scratch; that is, your program must perform a full grid-search from a random starting search point for w and b.



Any other specifications not following the above are invalid. While it is possible to design the API differently, for consistency you must follow the specification given above.