Assignment 4
BIOENG 145/245
Due Monday 2/26 11:59PM

*****PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!*****

1 Python Functions

In this section, you will implement several functions to brush up on your basic Python skills and become familiar with essential machine learning metrics and functions. **More detailed specifications of the inputs and outputs and example test cases are included in the code**. Please look at `q1.py` for this problem. Submit `q1.py` to Gradescope when completed, there are no written responses for this problem.

1.1 `accuracy_score`

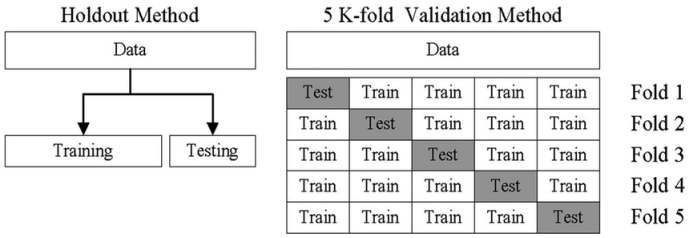
given the predictions (`y_pred`), calculate the accuracy score comparing to the actual answer (`y_true`). ex: `accuracy_score([1, 0, 1, 1, 0], [1, 0, 1, 0, 0]) = 0.80`; In layman terms, a function that performs what you do when you grade a multiple choice exam!

1.2 `train_test_split`

Create a function that shuffles the entire dataset, breaks it into training/test sets accordingly to the `test_size` given. Inputs and outputs specification in `q1.py`.

1.3 `k_fold_cv`

Shuffle (if `shuffle=True`) the dataset and divide X & y into k folds and do the following procedure. ex: partition X into $X_1, X_2, ..., X_k$ then, return a dataset of k items, where each X_i is the test set and the rest is the training set. In the example below, `k=5`.



1.4 `Normalize`

Normalize the dataset X along the feature dimension. Use this formula:

$$X_{\text{norm}} = \frac{X - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation.

2 Linear Gradient Descent

In this question, we will have you implement the Linear Gradient Descent class `LinearRegression`.

For linear regressions, we will use the mean-squared-error loss function:

$$\mathrm{L}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \quad \text{where} \quad \hat{y} = X\theta + b,$$

Here, θ is the weight vector and $\hat{y}$ is the predicted value by the model and y is the ground truth value. X is of shape (N, D) and θ is of shape (D, 1). N is the number of samples and D is the number of dimensions/features. Before you start to implement code, please use a pen and paper to derive the gradients:

$$\nabla_\theta \mathrm{L} = \frac{\partial \mathrm{L}}{\partial \theta} \quad \text{and} \quad \nabla_b \mathrm{L} = \frac{\partial \mathrm{L}}{\partial b}$$

Of course, with the gradients computed, we can simply update our weights and bias using the following form:

$$\theta = \theta - \eta * \nabla_\theta \mathrm{L} \quad \text{and} \quad b = b - \eta * \nabla_b \mathrm{L}$$

Functions for you to implement:

- `mean_squared_error(self, X, y)`: Takes in a data matrix `X` and the ground truth `y` and computes the mean squared error using class attributes `self.theta` and `self.bias`.
- `mean_squared_gradient(self, X_batch, y_batch)`: With the same inputs, this function outputs the gradient of the mean squared error loss function with respect to θ and b.
- `fit(self, X, y, epochs, lr, batch_size)`: This is the main function that will take in the data, labels, and some hyperparameters and run the actual gradient descent algorithm. Please note that your code will be tested to run batch, mini-batch, and stochastic gradient descents. (In this problem, by 'stochastic' gradient descent, we mean a gradient descent that considers the data-points one at a time.) Note: the total sample size may not be divisible by the batch size, so slight modifications may be needed.

For more specified details such as dimensions, inputs and outputs please check `q2.py`.

3 Submission

Here are the **2** files that need to be turned in to Gradescope:

- `q1.py`
- `q2.py`

For the coding portion, ***PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!*** The autograder will run your program and test the code, and whatever score you see after the autograder finishes running is the score you receive for the coding portion.