Raman Kahlon
PUID: 00291-51712

HW 3 Report

Part 2 Theory

1. In batch gradient descent, you have to go through all of the samples in the training set in order to perform a single update for a parameter in a specific iteration. Meanwhile, stochastic gradient descent uses either only one or a subset of the training samples from the training set in order to perform a single update for a parameter in a specific iteration. Due to this difference, if there is a large number of training samples, then stochastic gradient descent would be faster than batch gradient descent, since you are only using one training sample instead of running through the complete training set.

2. You would know that the model has converged when the loss function gets low enough that the change in gradient over each iteration is minuscule and insignificant.

3. The bias term is used in BGD/SGD learning to account for noise in the data.

4. True, because stochastic gradient descent uses only one or a subset of training samples from the training set to perform an update. Batch gradient descent iterates through all of the samples in the training set to perform an update. Therefore, stochastic gradient descent performs less computation per update than batch gradient descent.

5. We randomly shuffle the training examples before using SGD optimization because we need to make sure that the training set is representative of the overall distribution of the data. Thus, by shuffling the data, we can reduce the variance of the training set so that the models remain general and there is a lower chance of overfitting the data.

6. The hinge loss function is defined as f(x) = max(0, x) and is non-differentiable at 0. If we start at any random point and want to calculate the gradient, it will be highly unlikely to find the point of non-differentiability. Therefore, we need to rely on subgradient at the point of non-differentiability. This gives the gradient of hinge loss as:

$$\frac{\partial\, f(w)}{\partial x} = \frac{\partial\, max(0, x))}{\partial x} = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$$

7. A

8. Regularization is used to prevent overfitting of an algorithm in machine learning. It can also be used to steady the estimates during the collinearity present in the data. It can also be used. Regularization prevents overfitting because the algorithm would produce high coefficients for predictors. L_2 regularization represents when the probability of the coefficient decreases as it gets farther away from 0. Due to this, L_2 regularization could have issues when the regularization coefficient (lambda hyperparameter) is decreasing (taking negative values).

Part 3.1 Algorithm

Fit function:

    Randomize training data
    Split into classes and features vectors
    Loop through num_iter
        Initialize gradient vector to zeros
        Initialize gradient_bias variable to 0
        Loop through features vector
            Increase gradient vector by difference of classes and sigmoid of dot product of (weights, features) multiplied by features vector
            Increase gradient_bias variable by difference of classes and sigmoid of dot product of (weights, features)
        Increment weights vector by learning rate multiplied by gradient vector
        Increment gradient_bias by learning rate multiplied by gradient_bias value

Sigmoid function:

    Return result of sigmoid equation

Predict function:

    Loop through test_x
        If prediction <= 0
            Set prediction to -1
        Else
            Set prediction to 1
        Append prediction to output vector
    Return output vector

Part 4.1 Algorithm

Fit function:

- Randomize training data
- Split into classes and features vectors
- Loop through num_iter
  - Initialize gradient vector to zeros
  - Initialize gradient_bias variable to 0
  - Loop through features vector
    - Increase gradient vector by product of learning rate and gradient vector
    - Increase gradient_bias variable by product of learning rate and gradient_bias
  - Increment weights vector by learning rate multiplied by gradient vector
  - Increment gradient_bias by learning rate multiplied by gradient_bias

Predict function:

- Loop through test_x
  - If prediction = 0
    - Set prediction to -1
  - Append prediction to output vector
- Return output vector