3. Perceptron Algorithm

Extension Note: Project 1 due date has been extended by 2 days to **July 4 23:59UTC** (Note the UTC time zone).

Now you will implement the Perceptron algorithm

Perceptron Single Step Update

1.0/1 point (graded)

Now you will implement the single step update for the perceptron algorithm (implemented with $0-1$ loss). You will be given the feature vector as an array of numbers, the current θ and θ_0 parameters, and the correct label of the feature vector. The function should return a tuple in which the first element is the correctly updated value of θ and the second element is the correctly updated value of θ_0.

Available Functions: You have access to the NumPy python library as `np`.

Tip:: Because of numerical instabilities, it is preferable to identify 0 with a small range $[-\varepsilon, \varepsilon]$. That is, when x is a float, "$x = 0$" should be checked with $|x| < \varepsilon$.

```
            algorithm before this update.
        current_theta_0 - The current theta_0 being used by the perceptron
            algorithm before this update.

    Returns: A tuple where the first element is a numpy array with the value of
    theta after the current update has completed and the second element is a
    real valued number with the value of theta_0 after the current updated has
    completed.
    """
    epsilon = 10**(-10)
    if label * (np.dot(feature_vector, current_theta) + current_theta_0) < epsilon:
        current_theta += feature_vector * label
        current_theta_0 += label
    return (current_theta, current_theta_0)

```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def perceptron_single_step_update(
        feature_vector,
        label,
        current_theta,
        current_theta_0):
    """
    Properly updates the classification parameter, theta and theta_0, on a
    single step of the perceptron algorithm.

    Args:
        feature_vector - A numpy array describing a single data point.
        label - The correct classification of the feature vector.
        current_theta - The current theta being used by the perceptron
            algorithm before this update.
        current_theta_0 - The current theta_0 being used by the perceptron
            algorithm before this update.

    Returns: A tuple where the first element is a numpy array with the value of
    theta after the current update has completed and the second element is a
    real valued number with the value of theta_0 after the current updated has
    completed.
    """
    if label * (np.dot(current_theta, feature_vector) + current_theta_0) ≤ 0:
        current_theta += label * feature_vector
        current_theta_0 += label
    return (current_theta, current_theta_0)
```

Test results

See full output

```
CORRECT
```

See full output

Solution:

See above for expected answer.

We need to make sure that if the output of the perceptron is 0, the weights are still updated, hence the check for large inequality. In fact, because of numerical instabilities, it is preferable to identify 0 with a small range $[-\varepsilon, \varepsilon]$.

```
def perceptron_single_step_update(feature_vector, label, current_theta, current_theta_0):
        if label * (np.dot(current_theta, feature_vector) + current_theta_0) ≤ 1e-7:
                return (current_theta + label * feature_vector, current_theta_0 + label)
        return (current_theta, current_theta_0)
```

Submit You have used 4 of 20 attempts

Answers are displayed within the problem

Full Perceptron Algorithm

1.0/1 point (graded)
In this step you will implement the full perceptron algorithm. You will be given the same feature matrix and labels array as you were given in **The Complete Hinge Loss**. You will also be given T, the maximum number of times that you should iterate through the feature matrix before terminating the algorithm. Initialize θ and θ_0 to zero. This function should return a tuple in which the first element is the final value of θ and the second element is the value of θ_0.

Tip: Call the function `perceptron_single_step_update` directly without coding it again.

Hint: Make sure you initialize `theta` to a 1D array of shape `(n,)`, and **not** a 2D array of shape `(1, n)`.

Note: Please call `get_order(feature_matrix.shape[0])`, and use the ordering to iterate the feature matrix in each iteration. The ordering is specified due to grading purpose. In practice, people typically just randomly shuffle indices to do stochastic optimization.

Available Functions: You have access to the NumPy python library as `np` and `perceptron_single_step_update` which you have already implemented.

```

    Returns: A tuple where the first element is a numpy array with the value of
    theta, the linear classification parameter, after T iterations through the
    feature matrix and the second element is a real number with the value of
    theta_0, the offset classification parameter, after T iterations through
    the feature matrix.
    """
    n = feature_matrix.shape[1]
    theta = np.zeros(n)
    theta_0 = 0
    for t in range(T):
        for i in get_order(feature_matrix.shape[0]):
            theta, theta_0 = perceptron_single_step_update(feature_matrix[i], labels[i], theta, theta_0)
    return (theta, theta_0)

```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def perceptron(feature_matrix, labels, T):
    """
    Runs the full perceptron algorithm on a given set of data. Runs T
    iterations through the data set, there is no need to worry about
    stopping early.

    NOTE: Please use the previously implemented functions when applicable.
    Do not copy paste code from previous parts.

    Args:
        feature_matrix -  A numpy matrix describing the given data. Each row
            represents a single data point.
        labels - A numpy array where the kth element of the array is the
            correct classification of the kth row of the feature matrix.
        T - An integer indicating how many times the perceptron algorithm
            should iterate through the feature matrix.

    Returns: A tuple where the first element is a numpy array with the value of
    theta, the linear classification parameter, after T iterations through the
    feature matrix and the second element is a real number with the value of
    theta_0, the offset classification parameter, after T iterations through
    the feature matrix.
    """
    (nsamples, nfeatures) = feature_matrix.shape
    theta = np.zeros(nfeatures)
    theta_0 = 0.0
    for t in range(T):
        for i in get_order(nsamples):
            theta, theta_0 = perceptron_single_step_update(
                feature_matrix[i], labels[i], theta, theta_0)
    return (theta, theta_0)
```

Test results

See full output

```
CORRECT
```

See full output

Submit You have used 2 of 20 attempts

Answers are displayed within the problem

Average Perceptron Algorithm

1.0/1 point (graded)

The average perceptron will add a modification to the original perceptron algorithm: since the basic algorithm continues updating as the algorithm runs, nudging parameters in possibly conflicting directions, it is better to take an average of those parameters as the final answer. Every update of the algorithm is the same as before. The returned parameters θ, however, are an average of the θs across the nT

steps:

$$\theta_{final} = \frac{1}{nT}(\theta^{(1)} + \theta^{(2)} + \ldots + \theta^{(nT)})$$

You will now implement the average perceptron algorithm. This function should be constructed similarly to the Full Perceptron Algorithm above, except that it should return the average values of θ and θ_0

Tip: Tracking a moving average through loops is difficult, but tracking a sum through loops is simple.

Note: Please call `get_order(feature_matrix.shape[0])` , and use the ordering to iterate the feature matrix in each iteration. The ordering is specified due to grading purpose. In practice, people typically just randomly shuffle indices to do stochastic optimization.

Available Functions: You have access to the NumPy python library as `np` and `perceptron_single_step_update` which you have already implemented.

```
    find a sum and divide.
    """
    n_data = feature_matrix.shape[0]
    n_para = feature_matrix.shape[1]
    theta = np.zeros(n_para)
    theta_0 = 0
    theta_mean = np.zeros(n_para)
    theta_0_mean = 0
    for t in range(T):
        for i in get_order(n_data):
            theta, theta_0 = perceptron_single_step_update(feature_matrix[i], labels[i], theta, theta_0)
            theta_mean += theta/(T*n_data)
            theta_0_mean += theta_0/(T*n_data)
    return (theta_mean, theta_0_mean)

```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def average_perceptron(feature_matrix, labels, T):
    """
    Runs the average perceptron algorithm on a given set of data. Runs T
    iterations through the data set, there is no need to worry about
    stopping early.

    NOTE: Please use the previously implemented functions when applicable.
    Do not copy paste code from previous parts.

    Args:
        feature_matrix -  A numpy matrix describing the given data. Each row
            represents a single data point.
        labels - A numpy array where the kth element of the array is the
            correct classification of the kth row of the feature matrix.
        T - An integer indicating how many times the perceptron algorithm
            should iterate through the feature matrix.

    Returns: A tuple where the first element is a numpy array with the value of
    the average theta, the linear classification parameter, found after T
    iterations through the feature matrix and the second element is a real
    number with the value of the average theta_0, the offset classification
    parameter, found after T iterations through the feature matrix.

    Hint: It is difficult to keep a running average; however, it is simple to
    find a sum and divide.
    """
    (nsamples, nfeatures) = feature_matrix.shape
    theta = np.zeros(nfeatures)
    theta_sum = np.zeros(nfeatures)
    theta_0 = 0.0
    theta_0_sum = 0.0
    for t in range(T):
        for i in get_order(nsamples):
            theta, theta_0 = perceptron_single_step_update(
                feature_matrix[i], labels[i], theta, theta_0)
            theta_sum += theta
            theta_0_sum += theta_0
    return (theta_sum / (nsamples * T), theta_0_sum / (nsamples * T))
```

Test results