Lab Assignment 3: Implement Perceptron Learning Algorithm

Assignment 3.1

In this assignment, you will implement the perceptron model using different data set in python.

A perceptron is a fundamental unit of the neural network which takes weighted inputs, process it and capable of performing binary classifications.

Perceptron Recap

In the perceptron model inputs can be real numbers unlike the Boolean inputs in MP Neuron Model. The output from the model will still be binary {0, 1}. The perceptron model takes the input x if the weighted sum of the inputs is greater than threshold b output will be 1 else output will be 0.

$$\hat{y} = 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq b$$

$$\hat{y} = 0 \text{ otherwise}$$

Fig 1— Mathematical RepresentationLearning Algorithm

The main goal of the learning algorithm is to find vector **w** capable of absolutely separating Positive **P** (y = 1) and Negative **N**(y = 0) sets of data. Perceptron learning algorithm goes like this,

```
Algorithm: Perceptron Learning Algorithm
P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P$ and $\sum_{i=0}^{n} w_i * x_i < 0$ then
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N$ and $\sum_{i=0}^{n} w_i * x_i \geq 0$ then
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are
  classified correctly
```

Steps you can follow:

1. Before start building the Perceptron Model, first we need to load the required packages and the data set.
   There are a lot of linearly separable datasets available. You can select dataset of your choice.
2. Once we load the data, we need to grab the features X and response variables Y.
3. After fetching the X and Y variables, you will perform Min-Max scaling to bring all the features in the range 0—1.
4. Before building the model, split the data so that you can train the model on training data and test the performance of the model on testing data.
5. Initializes the weights vector **w** and threshold **b**
6. The function model takes input values **x** as an argument and perform the weighted aggregation of inputs (dot product between **w.x**) and returns the value 1 if the aggregation is greater than the threshold **b** else 0.
7. Define the predict function that takes input values **x** as an argument and for every observation present in **x**, the function calculates the predicted outcome and returns a list of predictions.
8. Finally, implement fit function to learn the best possible weight vector **w** and threshold value **b** for the given data.
9. The function takes input data(**x** & **y**), learning rate and the number of epochs as arguments.
10. Initialize a new perceptron class object and using that object call fit method on your training data to learn the best possible parameters.
11. Evaluate the model performance on the test data by calculating the testing accuracy.

Further Improvements

You can try out a few possible improvements to increase the accuracy of the model,

Vary the train-test size split and see if there is any change in accuracy.

Choose larger epochs values, learning rates and test on the perceptron model and visualize the change in accuracy.

Take random weights in the perceptron model and experiment.

Problems with Perceptron’s

Although the perceptron classified the two Iris flower classes perfectly, convergence is one of the biggest problems of the perceptron. Frank Rosenblatt proofed mathematically that the perceptron learning rule converges if the two classes can be separated by linear hyperplane, but problems arise if the classes cannot be separated perfectly by a linear classifier.

Even at a lower training rate, the perceptron failed to find a good decision boundary since one or more samples will always be misclassified in every epoch so that the learning rule never stops updating the weights.

Gradient Descent

Being a continuous function, one of the biggest advantages of the linear activation function over the unit step function is that it is differentiable. This property allows us to define a cost function J(w) that we can minimize in order to update our weights. In the case of the linear activation function, we can define the cost function J(w) as the sum of squared errors (SSE), which is similar to the cost function that is minimized in ordinary least squares (OLS) linear regression.

$$J(\mathbf{w}) = \frac{1}{2} \sum_i \left(\text{target}^{(i)} - \text{output}^{(i)}\right)^2 \qquad \text{output}^{(i)} \in \mathbb{R}$$

In order to minimize the SSE cost function, we will use gradient descent, a simple yet useful optimization algorithm that is often used in machine learning to find the local minimum of linear systems.

Assignment 2:

Builds the logistic regression model by calling the function you've implemented previously

Arguments:

X_train -- training set represented by a numpy array of shape (num_px * num_px * 3, m_train)

Y_train -- training labels represented by a numpy array (vector) of shape (1, m_train)

X_test -- test set represented by a numpy array of shape (num_px * num_px * 3, m_test)

Y_test -- test labels represented by a numpy array (vector) of shape (1, m_test)

num_iterations -- hyperparameter representing the number of iterations to optimize the parameters

learning_rate -- hyperparameter representing the learning rate used in the update rule of optimize()

print_cost -- Set to true to print the cost every 100 iterations

1. Build the general architecture of a learning algorithm, including:
    a. Initializing parameters
    b. Calculating the cost function and its gradient
    c. Using an optimization algorithm (gradient descent)
2. Gather all three functions above into a main model function, in the right order.