DS 5010 Homework 4

Instructions

- Submit your solutions on Canvas by the deadline displayed online.
- Your submission must include a single Python module (file with extension ".py") that includes all of the code necessary to answer the problems. All of your code should run without error.
- Problem numbers must be clearly marked with code comments. Problems must appear in order, but later problems may use functions defined in earlier problems.
- Functions must be documented with a docstring describing at least (1) the function's purpose, (2) any and all parameters, (3) the return value. Code should be commented such that its function is clear.
- All solutions to the given problems must be your own work. If you use third-party code for ancillary tasks, you **must** cite them. (You may use code from class notes without citation.)
- You may use functions from built-in modules (e.g., `math`). You may **NOT** use external modules (e.g., `numpy`, `pandas`, etc.).

In this assignment, you will implement a simple supervised learning classifier called the perceptron. The perceptron is the basis for neural networks, where it serves as an artificial neuron. You may use the code from "hw4-skeleton.py" on Piazza as a starting point.

The `Perceptron` class is initialized with a single attribute `weights`, which is a list of weights that will be updated through training examples of input/output pairs. The `__init__()` method is already defined, and (optionally) allows the user to provide a list of initial weights.

You can also find an example dataset called "sonar.csv" on Piazza for testing your classifier on real data.

Problem 1 A perceptron predicts output by calculating the weighted sum of the input. If we have an input sample $x = [1, x_1, x_2, ..., x_m]$ and weights $w = [w_0, w_1, ..., w_m]$, then we calculate the *activation* as:

$$activation = \sum_{i=0}^{m} w_i x_i \tag{1}$$

Then we predict the output using the following rule:

$$\begin{aligned} &f(x) = 1 \; if \; activation > 0, \\ &f(x) = 0 \; otherwise \end{aligned} \tag{2}$$

The term "activation" refers to the metaphor of the perceptron as an artifical neuron. We classify the sample as positive (1) if the activation is over a certain threshold, and negative (0) otherwise.

Note that we always prepend the input vector $x = [1, x_1, x_2, ..., x_m]$ with $x_0 = 1$; the corresponding weight w_0 is called the *bias*, and serves a role similar to the *intercept* in the equation for a line.

As an example, consider an input $x = [-1, -1]$ and a perceptron with weights $w = [-5, 1, 1]$. We would calculate the activation as:

$$\begin{aligned} activation &= (-5)(1) + (1)(-1) + (1)(-1) \\ &= -7 \end{aligned} \tag{3}$$

Since $activation < 0$, we would predict 0.

Define the method `Perceptron.predict1(self, x)` satisfying the following criteria:

- Takes parameter `x` which is a single input sample provided as a list or tuple
- Returns the predicted output (0 or 1) from the single input sample using the current weights
- Prints a message that the "model has not been trained yet" if the weights are `None`

Examples:

```
In : X = [(-1, -1),
        (-5, -2.5),
        (-7.5, -7.5),
        (10, 7.5),
        (-2.5, 12.5),
        (5, 10),
        (5, 5)]

In : Y = [0, 0, 0, 1, 0, 1, 1]

In : model = Perceptron(weights=[-5, 1, 1])

In : model.predict1(X[0]) # correct
Out: 0

In : model.predict1(X[1]) # correct
Out: 0

In : model.predict1(X[2]) # correct
Out: 0

In : model.predict1(X[3]) # correct
Out: 1

In : model.predict1(X[4]) # incorrect!
Out: 1
```

Problem 2 Define the method `Perceptron.predict(self, X)` satisfying the following criteria:

- Takes parameter `X` which is an iterable of multiple input samples (each being a list or tuple)
- Returns a list of all predicted outputs (0 or 1) from the input samples using the current weights

Hint: Use your method from Problem 1.

Examples:

```
In : X = [(-1, -1),
        (-5, -2.5),
        (-7.5, -7.5),
        (10, 7.5),
        (-2.5, 12.5),
        (5, 10),
        (5, 5)]

In : Y = [0, 0, 0, 1, 0, 1, 1]

In : model = Perceptron(weights=[-5, 1, 1])

In : model.predict(X) # all correct but X[4]
Out: [0, 0, 0, 1, 1, 1, 1]
```

Problem 3 The perceptron follows a simple "online" update rule that re-calculates the weights from each input/output example it receives during training. If we have an input sample $x = [1, x_1, x_2, ..., x_m]$ with known output y, then we calculate the predicted output y_{pred}.

If the prediction is correct ($y_{pred} = y$), then we do nothing, and the weights are not updated.

If the prediction is incorrect ($y_{pred} \neq y$), then we update the weights $w = [w_0, w_1, ..., w_m]$ using the rule:

$$w_{i,new} = w_{i,old} + (y - y_{pred})x_i \tag{4}$$

for all features $i = 0, ..., m$.

This update shifts the weights (and therefore the decision boundary) toward correctly classifying x. To fully train the model, we simply apply this update rule to all input/output samples. Usually, we will loop through the full dataset multiple times to iteratively improve the model further after each round of updates.

Define the method `Perceptron.update(self, x, y)` satisfying the following criteria:

- Take parameter `x` which is a single input sample provided as a list or tuple
- Take parameter `y` which is a single output sample (0 or 1)
- Initializes the weights to a list of 0's if they are still `None`
- Performs a single training update using the sample input/output pair

Examples:

```
In : X = [(-1, -1),
        (-5, -2.5),
        (-7.5, -7.5),
        (10, 7.5),
        (-2.5, 12.5),
        (5, 10),
        (5, 5)]

In : Y = [0, 0, 0, 1, 0, 1, 1]

In : model = Perceptron(weights=[-5, 1, 1])

In : model.update(X[4], Y[4])

In : print(model.weights) # updated weights
[-6, 3.5, -11.5]

In : model.predict1(X[4]) # X[4] predicted correctly now!
Out: 0

In : model.predict(X) # others are wrong now though
Out: [1, 1, 1, 0, 0, 0, 0]
```

Problem 4 Define the method `Perceptron.fit(self, X, Y, num_iter = 100)` satisfying the following criteria:

- Takes parameter `X` which is an iterable of input samples (each being a list or tuple)
- Take parameter `Y` which is an iterable of output samples (each 0 or 1)
- (A) Loops through all sample input/output pairs performing a training update for each pair
- (B) Repeats the training loop (A) for as many iterations as specified by `num_iter`.

Hint: Use your method from Problem 3.

Examples:

```
In : X = [(-1, -1),
        (-5, -2.5),
        (-7.5, -7.5),
        (10, 7.5),
        (-2.5, 12.5),
        (5, 10),
        (5, 5)]

In : Y = [0, 0, 0, 1, 0, 1, 1]

In : model = Perceptron()

In : model.fit(X, Y, num_iter=100)

In : print(model.weights) # fitted weights
[0, 12.5, -5.0]

In : model.predict(X) # perfectly separable
Out: [0, 0, 0, 1, 0, 1, 1]
```

Problem 5 Define the method `Perceptron.score(self, X, Y)` satisfying the following criteria:

- Takes parameter `X` which is an iterable of input samples (each being a list or tuple)
- Take parameter `Y` which is an iterable of output samples (each 0 or 1)
- Returns the accuracy of predicting `Y` from `X` using the current weights
- Accuracy is calculated as the proportion of correct predictions

Examples:

```
In : X = [(-1, -1),
        (-5, -2.5),
        (-7.5, -7.5),
        (10, 7.5),
        (-2.5, 12.5),
        (5, 10),
        (5, 5)]

In : Y = [0, 0, 0, 1, 0, 1, 1]

In : model = Perceptron(weights=[-5, 1, 1])

In : model.score(X, Y)
Out: 0.8571428571428571
```