INFO 4604/5604, Fall 2020
Assignment #1: Perceptron (Version 1.02)
Due by **11:59pm** on **Friday, September 18**

---

1 Instructions

1.1 What You are Given

The file hw1.ipynb is available under homework assignment #1 on Canvas. This notebook is adapted from code from *Python Machine Learning*.

1.2 What to Hand In

You will turn in a completed notebook on Canvas. Your submission should be contained in a single Jupyter notebook named `hw1-lastname.ipynb`, where *lastname* is replaced with your last name. This will include your code as well as figures and answers to questions. Please create headers for your cells that are labeled "Deliverable" followed by the problem number, so that your responses are easy to find.

Note on 4604 vs 5604: Sections labeled as [**5604**] are only required for INFO-5604 students. Students in INFO-4604 may optionally do these problems for extra credit (with the amount of extra credit points indicated as "EC" points).

1.3 Submission Policies

- **Collaboration:** You are allowed to work with a partner. You are still expected to write up your own solution. Each individual must post their own submission on Canvas, and you must list the names of your partner in your notebook.
- **Late Submissions:** We allow each student to use up to 5 late days over the semester. You have late days, not late hours. This means that if your submission is late by any amount of time past the deadline, then this will use up a late day. If it is late by any amount beyond 24 hours past the deadline, then this will use a second late, and so on. Once you have used up all late days, late assignments will not receive credit except in special circumstances.

1.4 Asking for Help

You are encouraged to ask questions on Canvas. Do not post anything that you are turning in. In this assignment, that would be any of the plots you need to hand in, or the parameter values. However, you can describe your results, like the number of iterations it took to converge, and general things you observe about the algorithms.

You may ask questions to help with debugging, but do not post your code. You can share error messages and describe what you are trying to do with your code, but try not to write enough to "give away" the solution.

2 Experimenting with Perceptron

The code from the book uses two variables, *sepal length* and *petal length* (columns 1 and 3). This is not actually a very interesting example, because the points are completely separated by only one of these dimensions (*petal length*), so it is easy to learn. In other words, if you only used this one feature, you could still learn a classifier, rather than needing two features.

A more interesting example is with *sepal length* and *sepal width* The data are still linearly separable in these two dimensions, but it requires a combination of both variables. Modify the code to use these two variables. When you run it, you will find that the algorithm still makes errors after the default 10 iterations (also called epochs). Change the `n_iters` parameter of the `Perceptron` object to 1000. You will see that eventually perceptron learns to classify all of the instances correctly, but that it takes a large number of iterations.

Examine the figure of decision regions (the figure with two regions shaded as red or blue) when running perceptron for 10, 20, 50, 100, 200, 500, and 1000 iterations (seven different figures). You can see how the boundary changes and gradually becomes more accurate as the algorithm runs for more iterations.

In your figures, be sure to change the axis labels to reflect the correct variable name (*sepal width* instead of *pedal length*).

Finally, examine the weights that are learned by perceptron after 1000 iterations. To do this, print out the values of the `w_` variable of the `Perceptron` object, which is the weight vector, where `w_[0]` is the 'bias' weight.

2.1 Deliverables [8 points]

Include the following in your writeup:

1. The plot showing the number of updates versus epochs when using 1000 iterations.
2. The seven region boundary figures for different numbers of iterations.
3. Write the linear function that is learned, in the form $m_1x_1 + m_2x_2 + b$, based on the parameters in the `w_` vector as described above.

3 Modifying Perceptron [5604]

You will experiment with an algorithm that is similar to perceptron, called **Winnow**. Winnow is not a commonly used algorithm (and it's not quite appropriate for this dataset), but implementing it is a way to get additional practice with the perceptron implementation here.

Like perceptron, Winnow (at least the basic version) is used for binary classification, where the positive class is predicted if:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} \geq \tau$$

for some threshold τ. This is almost the same as the perceptron prediction function, except that in perceptron, the threshold τ for positive classification is usually 0, while it must be positive in Winnow. Usually τ is set to the number of training instances in Winnow. In this assignment, you'll set $\tau = 100.0$.

The main difference between Winnow and Perceptron is how the weights are updated. Perceptron uses additive updates while Winnow uses multiplicative updates. The Winnow update rule is:

$$w_i^{(\text{new})} = \begin{cases} w_i \times \eta x_i & y_i > f(x_i) \\ w_i \div \eta x_i & y_i < f(x_i) \\ w_i & y_i = f(x_i) \end{cases}$$

For this problem, you should create a new class called `Winnow`, which is similar to the `Perceptron` class but with the following changes:

- Initialize the weights **w** to 1.0 instead of 0.0. This can be done in `numpy` with `np.ones` (where the current code for the Perceptron uses `np.zeros`).
- Change the prediction function so that it outputs the positive class if the linear function is greater than 100.0 (where the current code for the Perceptron uses a threshold of 0.0).
- Change the update rule so that the weights are multiplied by ηx_i if the prediction was an underestimate, and divided by ηx_i if the prediction was an overestimate. The Winnow update rule (above) specifies what to implement. Note that the current code for the Perceptron adds ηx_i to the weight if it was an underestimate and subtracts ηx_i if it was an overestimate).

When you run this, use the variables *sepal length* and *petal length*. (The algorithm won't work if you use *sepal length* and *sepal width*, for reasons we won't get into.)

In the code, set the parameter `eta` to 1.0 (it won't behave as expected if $\eta < 1$, which is the default, so you need to change it), and set `n_iters` to 10.

You may notice that when η (eta) is 1, the bias weight will never change. This is okay; the algorithm will still work, but the bias won't contribute anything to the prediction rule.

If you run everything correctly, it should converge after 5 epochs.

3.1 Deliverables [5604: 6 points; 4604: +3 EC points]

Include the following in your writeup:

1. The plot showing the number of updates versus epochs when using 10 iterations.
2. The region boundary figure.
3. Write the linear function that is learned, in the form $m_1 x_1 + m_2 x_2 + b$.