Danny North

1/26/15

CS 478

Dr. Martinez

Perceptron Lab

I. The Learning Algorithm

For the perceptron learning algorithm, I created a class "Perceptron" that took as arguments; a threshold, learning rate, and bias. Internally, the class holds as members; threshold (θ) , weights (w), bias, and learning rate (c). For convenience for testing I also included a delta weight variable, which is the weight change after one input is given.

After creating an array the size of the inputs (x) + 1 for the bias, and reading in the inputs and targets (t) for training, the Perceptron uses the equation:

$$total = \sum_{i=1}^{n} x_i w_i$$

to give an output (z) prediction for the target. The prediction is based on the following rules:

$$1 \; if \; total \geq \; \theta$$

$$0 \; if \; total < \; \theta$$

if the output matched the target, then the perceptron will be “satisfied” and move on to the next input. If not, then the perceptron will “learn” by updating each weight based on the following equation:

$$\Delta w_i = c * (t - z) * x_i$$

$$w_i += \; \Delta w_i$$

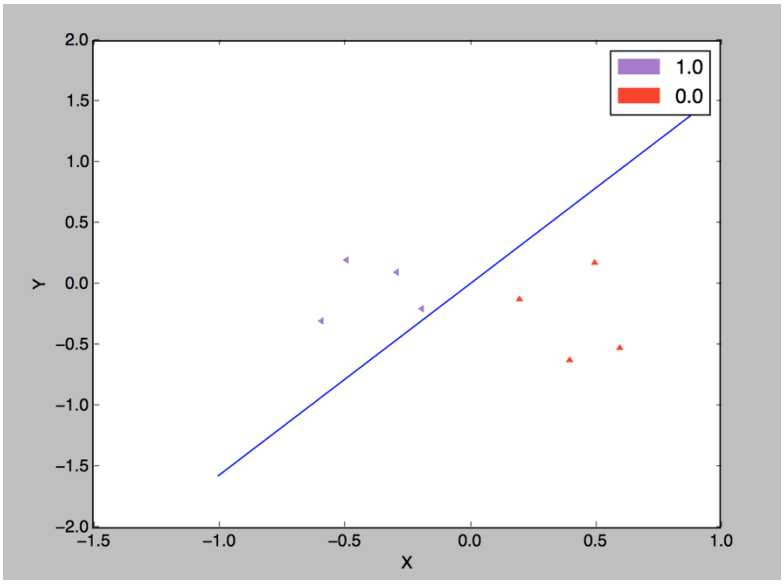
In order to try and accommodate for a good model, the training algorithm is designed to stop when there is no weight change after three consecutive epochs, or it has reached 1000 epochs. This is to make sure we’re not missing any interesting changes that could occur between epochs, but also not trying to fine-tune infinitely. The 1000 epoch limit is especially helpful for non-linearly separable problems because it limits time the algorithm is running for.

II. **Training on the created ARFFs**

For the creation of the two ARFF files, I used the linearlySeparable.arff as an example file for my first one and used the same points for the second one, but changed the targets so that it would become non-linearly separable. Below are the results of these tests, including a confusion matrix and graphs for both tests:

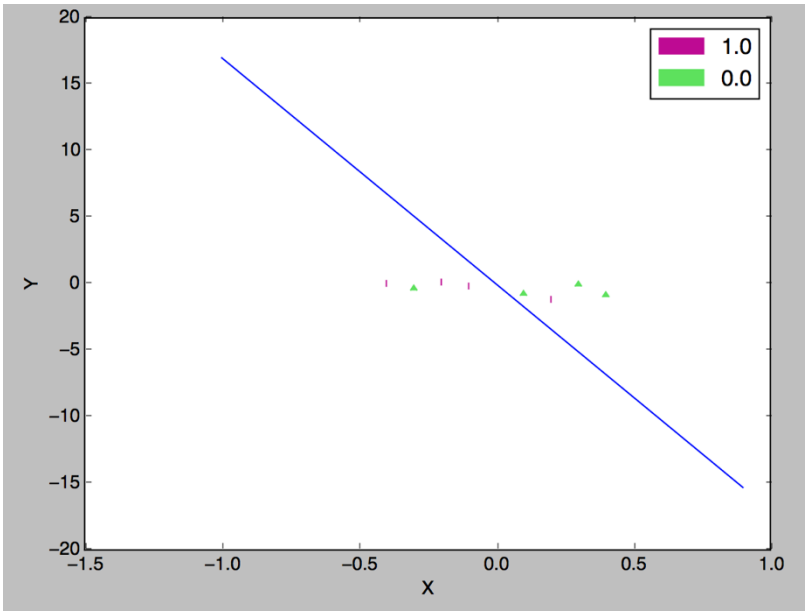
a. Linearly Separable Test

```
Global Change:  [-0.11  0.07  0.]
Trained after  4  Epochs.
accuracy: 100.000%

[[ 4.  0.]
 [ 0.  4.]]
```



b. Non Linearly Separable Test

```
Global Change:  [ -3.50000000e-01
                  -4.35068648e-15
                  0.00000000e+00]
Trained after  1000  Epochs.
accuracy: 75.000%

[[ 3.  1.]
 [ 1.  3.]]
```



I also tested these two data sets with different learning rates (0.5, 1, 2, 0.01, .0001) and

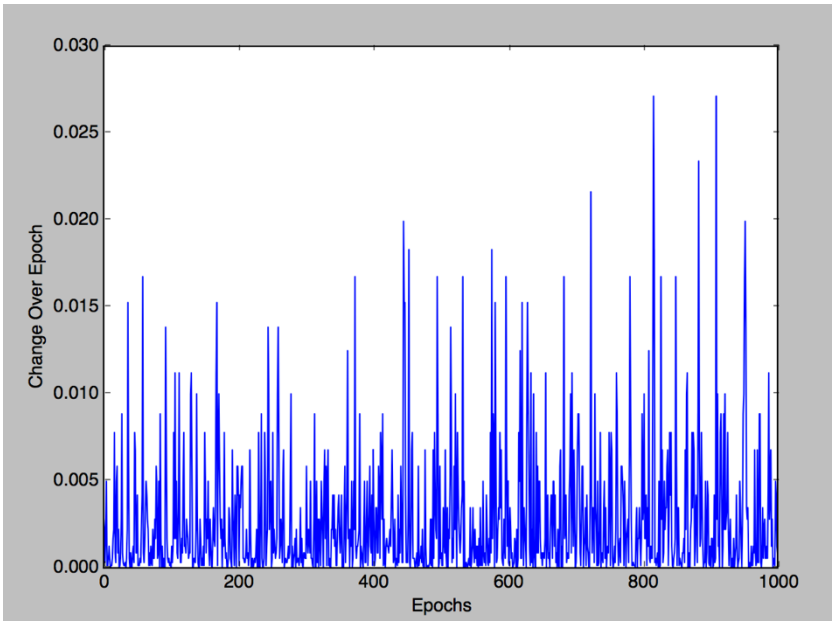
saw very little change. The only noticeable change I saw was the weight coefficients were different (but they still ended up with the same equation) and it would take 1 less epoch for larger learning rates, and 1 more epoch on occasion for very very small learning rates.

III. **Learning the Voting Task**

Here are the results of learning the voting task. Note that because of the requirement of either 0 change for 3 consecutive epochs, or 1000 epochs max, my algorithm ends up taking all 1000 epochs because it cannot fully settle (as seen from the graph). I'm sure there could be some fine tuning done to make it settle earlier, but for the purposes of this lab, the time it takes to compute does not make much of a difference.

```
Trained after  1000  Epochs.
accuracy: 98.551%
[[ 89.   1.]
 [  1.  47.]]

Trained after  1000  Epochs.
accuracy: 91.304%
[[ 81.   6.]
 [  6.  45.]]

Trained after  1000  Epochs.
accuracy: 94.203%
[[ 70.   5.]
 [  3.  60.]]

Trained after  1000  Epochs.
accuracy: 94.203%
[[ 74.   3.]
 [  5.  56.]]

Trained after  1000  Epochs.
accuracy: 94.203%
[[ 75.   4.]
 [  4.  55.]]
```

Average Accuracy: 93.29%
Average # Epochs: 1000



Here is also the global weight change from one of the tests:

```
Global Change:  [-0.5 -0.4 -1.1  3.3  1.5 -0.2  1.3  1.5 -1.6  1.  -1.4 -0.2 -0.1  0.2 -0.6]
```

We know that the weights furthest from 0 are the most critical. In this test, if the change is positive then it's more likely for a Republican, and if it's negative then it's more likely for a Democrat. So in this case, the 4th weight seems to be the most critical and seems to say that if you answered "yes" to the "physician-fee-freeze" then you are most likely Republican. In the same light, answering "yes" to "mx-missile" means that you are more likely to be Democrat. Other factors such as "crime" or "duty free exports" seem to give less information based on how you answer.

IV. **Experimentation**

For the experimentation part of this lab I decided to try and test the iris data set. In order for this to work properly I needed to change a lot of things. First, I added a "specifiedTarget" variable to the basic perceptron class. If the specified target existed, then the targets for the perceptron would change from the original three-class target (in the case of the iris, 0, 1, 2) to either 1 if the target matched the specifiedTarget, or 0 if it did not. This way, the problem was converted to a binary problem PER perceptron.

Then I created three perceptrons, one for each class, and gave each their specified target (0 for the setosa perceptron, 1 for the versicolor, and 2 for the virginica). Each Perceptron was trained based on that scheme, so that the weight vectors were positively correlated to its goal.

After this was setup, I changed the test algorithm to accommodate any number of Perceptrons. If the perceptron that was being tested returned a 1, it would be added to a dictionary of potential candidates for final prediction. The final prediction was chosen based on which net value was the highest if more than one potential candidate existed. The experiment turned out very well and accuracy ranged from 67% - 95%, dependent on the random factor involved. Below are some of the results, including a Confusion Matrix for clarity. (It should also be noted that the order is Setosa, Versicolor, and Virginica).

```
Global Change:  [ 0.17  0.34 -0.61 -0.3   0.1 ]
Trained after  4  Epochs.
Global Change:  [ -0.46  -6.28   3.66  -7.61  17.3 ]
Trained after  1000  Epochs.
Global Change:  [ -6.66  -8.87  13.72  19.71 -27.3 ]
Trained after  1000  Epochs.
accuracy: 92.667%

[[ 49.   1.   0.]
 [  5.  40.   5.]
 [  0.   0.  50.]]


Global Change:  [ 0.17  0.4  -0.74 -0.34  0.1 ]
Trained after  4  Epochs.
Global Change:  [ -0.36  -6.55   2.96  -7.2   17.1 ]
Trained after  1000  Epochs.
Global Change:  [ -7.68  -9.16  12.72  19.45 -27.5 ]
Trained after  1000  Epochs.
accuracy: 72.000%

[[ 49.   1.   0.]
 [ 28.  22.   0.]
 [  9.   4.  37.]]
```

In order to try and “have fun” with this data set, I decided to try different learning rates on all three algorithms. I saw no effect other than the weight changes, but the accuracy seemed to stay the same, based on the random factor. It was…. Uneventful.