Charles Justin Reusnow

Professor Chakraborty

CMP SCI 4340

14 September 2017

Project #1

For my project, I choose to implement the Perceptron Learning Algorithm (PLA) in JavaScript so I could utilize the canvas API from HTML 5 to display the work in a more easily digestible fashion than just straight numbers. I consider myself a visual learner, and this proved to be very helpful in that regard. But because of this stipulation, I had to do a lot more coding to get the display to work right, so the majority of my code is irrelevant to the perceptron learning algorithm, but rather is overhead for doing things like fetching input data, displaying data on the canvas, and updating details in real time (like the iteration counter). Because of these added complexities, I feel it would be best if I explained a little how my interface works before letting you dive into it. And to preface, I went way overboard on this project and added a lot of features that were not requested in the specification. I understand that this was not requested, and do not expect to earn extra points because of it, I simply wanted to get a very good understanding of how the algorithm works and thus wanted to give myself as many tools as possible to explore it. In addition, I would like to note that the inputs on this project expect valid data to be entered without checking that it is valid. For example, if you type “a” under “maximum iterations” instead of a number, you will get unexpected behavior that was not handled.

I took the liberty of adding a screenshot of what my interface looks like (Figure 1). It isn’t the prettiest interface in terms of layout, but I believe I have everything segmented nicely enough to understand the purpose of each section.

Firstly, we have the PLA Simulator section (dark blue) which contains a canvas element that updates in real time to display points on the graph, the bounding equation line, axis limits, and axes. While the simulation is running, a line representing the bounding equation is displayed in red on the canvas. Once a bounding equation has been decided on by the PLA, the line is displayed once more but now as blue, and it will not move again. Below it is the action button which updates based on what step of the algorithm we are in. Below that is a status update section that tells the user what the algorithm is doing currently.

Next, we have the parameters section (pink), which is broken down further into subsections. The first subsection is the classes segment (green). This section is used to define what color and symbols each of the two classes of points should have. It can be any character or set of text for symbol, and any valid color for the color of the point. By default they are • for simplicity and clarity.

The next section is the data section (yellow) which contains a radio selection for what method of data input you would like. There are 4 options, preset, manual, file, and random. When preset is selected, you are given the option to select a preset data I crafted that I consider to be interesting, or relevant to the project. Please note that the first option under preset is compliant with the project specification (25 class A, 25 class B, and 30 test vectors). When manual is selected you are given a blank textarea element to fill in with a string of text. When using this option you must enter a string that conforms to a specific regular expression (see Appendix A for in depth explanation). The third option, File, is for loading in a .txt file containing a string that matches the regular expression described previously. This option is used for if you wrote a separate program that would output data into a text file and you wanted to load it in without copying and pasting the whole thing into the manual section. The fourth option is random, and it has a few options associated with it. You can specify the quantity of class A vectors, class B vectors, and test vectors. You can specify the ranges in which they appear on both axes. You can choose if you want the randomly generated data to be linearly separable or if you would rather them be completely randomly placed. If you choose it to be linearly separable you can specify which line they should be separable along (in slope-intercept form).

The next section we have is the constants section (orange). This section contains: Max Training Iterations – How many iterations the algorithm is allowed to run during training before terminating. Learning Rate – The amount by which the algorithm corrects for errors. Theta – Used for classifying points. Speed – The amount of time, in milliseconds, between each step of the algorithm. Initial Weight X, Y, and Bias – The initial weights for these features. Randomize Initial Weights – Whether the weights should be specified by the user or randomized.

The only other piece in the parameters section is the “Choose This Configuration” button which must be pressed before being able to run the algorithm, and also any changes made in the inputs will not be reflected in the simulator until this button is pressed.

Below this is the next section, the Details section (red). This section updates while the algorithm is running to give details on the state of the algorithm and plane. It contains: Training Iteration – What iteration of the training phase we are on. Testing Iteration – What iteration of the testing phase we are on. Bounding Equation – The current bounding equation for classifying test points. Weight X Y and Bias – The current calculated weights for those three features. Class A and B Vectors – How many vectors on the plane are currently in what class, this includes classified testing vectors. Classified Vectors – Updates to show how many test vectors have been classified compared to the total.

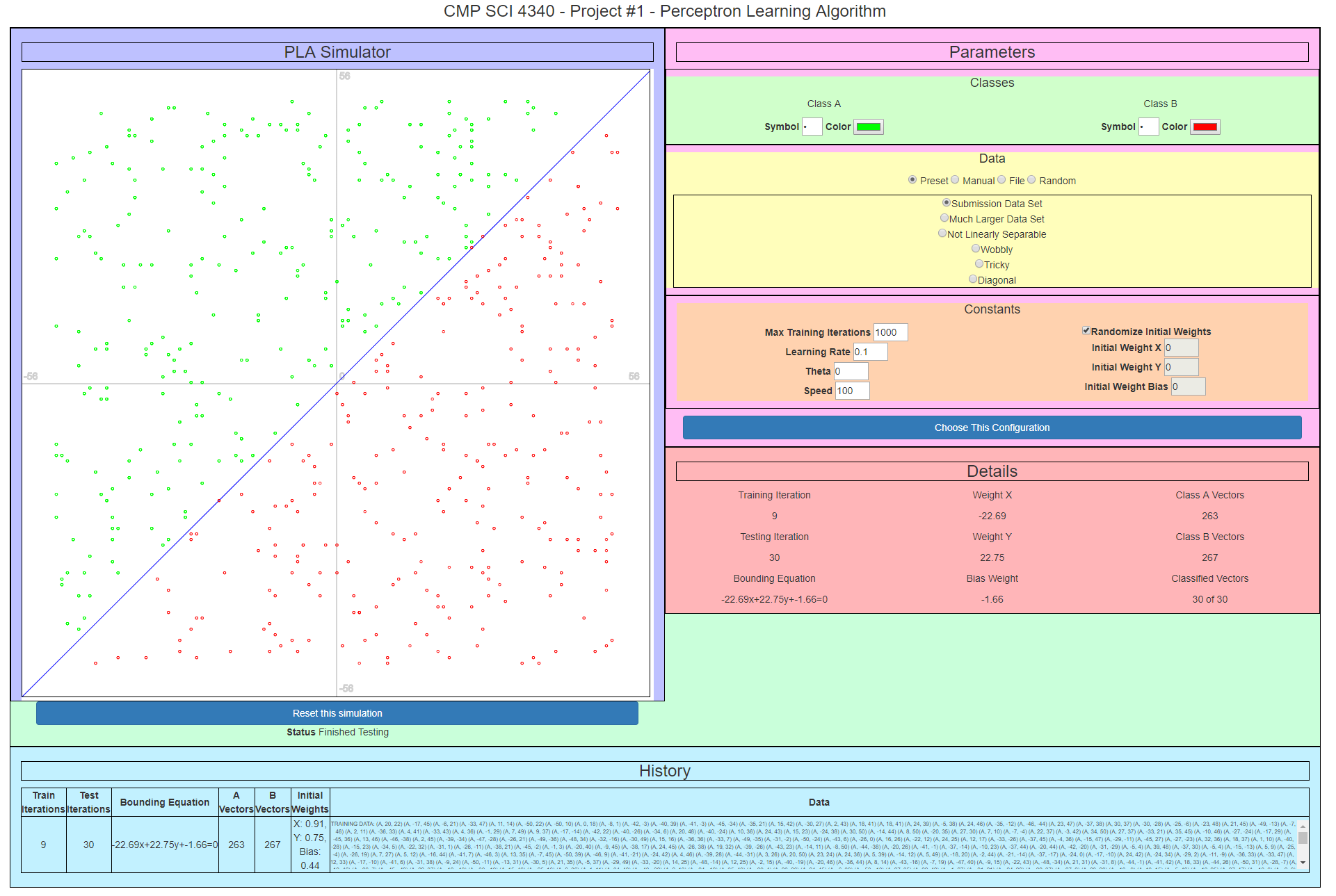
Below this is the final section, History (light blue). Upon completion of a run of the PLA in both training and test phases, the results are captured and added to this table. The only portion which may need explaining is the data section, this shows the input data regular expression (appendix A) compliant version of the data that was displayed during that run of the PLA.

Now that I have introduced you to the basics, I can answer the questions laid out in the project specification. Also, this is the exact set of data points used in my submission data set, any reference to “data” in the following answers refers to this dataset (preset 1):

TRAINING DATA: (A, -43, 35) (A, 12, 46) (A, -36, 26) (A, 49, 2) (A, -40, 22) (A, 43, 35) (A, -5, 30) (A, -38, 6) (A, 7, 23) (A, 31, 35) (A, -43, 37) (A, -23, 37) (A, -48, 47) (A, 21, 36) (A, 5, 32) (A, 19, 10) (A, -37, 16) (A, 7, 20) (A, 30, 18) (A, -5, 28) (A, 11, 21) (A, -50, 44) (A, 47, 14) (A, 3, 20) (A, -46, 39) (B, 2, -35) (B, 33, -29) (B, 36, -48) (B, -32, -21) (B, 38, -2) (B, 27, -28) (B, -36, -4) (B, -22, -34) (B, -40, -33) (B, 3, -33) (B, -41, -44) (B, 2, -6) (B, -21, -36) (B, 47, -11) (B, -11, -48) (B, -26, -29) (B, -17, -50) (B, 12, -23) (B, -22, -17) (B, -46, -3) (B, -29, -13) (B, -11, -28) (B, 36, -14) (B, 40, -19) (B, -11, -44); TESTING DATA: (-12, -50) (15, -3) (26, -33) (36, -17) (-38, -46) (9, -19) (-43, -19) (20, 8) (37, -30) (-28, -6) (-44, 25) (-46, -42) (-45, 49) (-30, 5) (-48, -7) (-39, 30) (-50, 15) (-10, -46) (50, 39) (47, -32) (-27, 42) (14, 41) (34, -29) (31, -5) (-9, -4) (-38, 28) (-5, -7) (6, 39) (-46, 42) (-2, -10).

1. The training data points in this set are linearly separable. All points of class A can be divided from all points of class B by a straight line.
2. The test points are certainly linearly separable, for they have no classification until after training is completed and they are interpreted based on their X and Y features (perhaps I am misunderstanding how the testing phase works? I thought this data is unclassified because it depends on the bounding equation found from the training data in order to classify it).
3. My initial choice of weights and constants are the default specified in the input, Max Iterations: 1000, Learning Rate: 0.1, Theta: 0, and the weights are randomized between 0 and 1 by default. For the sake of keeping things repeatable, I will run any simulations using all weights initialized to 0, unless otherwise stated.
4. The final solution equation of the line for the data specified and all constants as described previously is: 0.3x+7.1y-0.3=0.
5. The total number of weight vector updates my algorithm made is: 600.
6. The total number of iterations made over the training set is 3.
7. The final misclassification error is 0. The test data cannot have any misclassification error, they do not have a classification associated with them until after the algorithm is complete (again unless I am misunderstanding, which I now think I am).
8. To vary the weights, I first start with all weights at 0, then all weights at 1, then all weights at 10, then all weights at 100, here are my findings:
   1. 0: No change, operates exactly as before.
   2. 1: The training phase took 6 more iterations (9 total), but it still found a solution.
   3. 10: The training phase took only 7 iterations (2 less than with weights all at 1).
   4. 100: The training phase took 9 iterations total once again, but interestingly (though, not surprisingly), found a different solution from all weights at 1 which also took 9 total iterations.
9. To vary the learning rate, I will start at the default 0.1, try moving down to 0, go up to 0.5, up to 1, and then up again to 10.
   1. 0.1: No change, operates exactly as before.
   2. 0: Before running this one I hypothesized disaster, because without a learning rate with any real value, no progress can be made. This is exactly what happened, and the line stayed exactly at y=0 until it reached maximum iteration.
   3. 0.5: The solution was still found in the same number of iterations as the default, however the line found is different.
   4. 1: The solution was still found in the same number of iterations as the default, however the line found is different, and different from the previous run as well.
   5. 10: Same story as before. Though I am noticing a pattern. Each of the weights is equal to n \* the weight found in the default, where n = current learning rate / default learning rate. By that I mean, the weights found in this run are exactly 100 times larger than the weights found in the default run. I don’t think I will find any different results on this data set.
10. To vary the consideration of the data points, I will take the default data and rearrange a few data points, specifically the first A point will be swapped with the last A point, and the first B point will be swapped with the last B point.
    1. Original: No change, 3 iterations and 600 weight updates with no misclassification error and decision boundary 0.3x+7.1y-0.3=0.
    2. A & B swap: Interestingly, it took 7 iterations to solve it this time, 1200 weight updates, no misclassification error and decision boundary 0.6x+11.4y-0.9=0.

Figure 1 – The PLA Interface:



Appendix A – The Data Input Regular Expression

The data input regular expression is used to validate input strings are parse-able then parse them into their subordinate parts. The regular expression is:

/^ \*TRAIN(ING)? \*(DATA)? \*: \*((\( \*[AB] \*, \*-?\d+ \*, \*-?\d+ \*\) \*)+) \*; \*TEST(ING)? \*(DATA)? \*: \*((\( \*-?\d+ \*, \*-?\d+ \*\) \*)+) \*$/g

To start, there are some recurring themes in this regular expression. The first of which is: “ \*”. This combo appears many times, and it basically translates into “Anything between the previous regular expression item and the next regular expression item can be separated by any number of spaces including none.” In the interest of brevity, I left out an explanation of this item in the following page when the Regular Expression is broken down. Another common piece is: “-?\d+” Which translates into “Any integer, positive or negative, including zero”.

The high level version of what types of strings this regular expression is looking for are strings that follow this pattern: “TRAINING DATA: ([A or B], [integer], [integer])… ; TESTING DATA: ([integer], [integer])”. Note that the items in the parenthesized segments are comma separated, but the parenthesized segments themselves are not comma separated. Using capturing groups, we can extract the relevant information from the string and use it to populate our algorithm. See the next page for more details on the regular expression.

Here is the regular expression broken down piece by piece so it makes more sense:

/ - Begin the regular expression.

^ - Assert that the match must begin at the start of the string.

TRAIN – Matches the phrase “TRAIN” literally.

( - Begin first capturing group.

ING – Matches the phrase “ING” literally.

) – End first capturing group.

? – Matches the previous group zero or one times (its optional).

( - Begin second capturing group.

DATA – Matched the phrase “DATA” literally.

) – End second capturing group.

? – Matches the previous group zero or one times.

: - Matches “:” literally.

( - Begin third capturing group.

( - Begin fourth capturing group.

\( - Matches “(“ literally.

[AB], – Matches “A” or “B” literally, followed by “,”.

-?\d+ - Any integer positive or negative including zero.

, – Matches “,” literally.

-?\d+ - Any integer positive or negative including zero.

\) – Matches “)” literally.

) – End fourth capturing group

+ - Match the previous group 1 or more times.

) – End third capturing group.

; - Matches “;” literally.

TEST – Matches the phrase “TEST” literally.

( - Begin fifth capturing group.

ING – Matches the phrase “ING” literally.

) – End fifth capturing group.

? – Matches the previous group zero or one times (its optional).

( - Begin sixth capturing group.

DATA – Matched the phrase “DATA” literally.

) – End sixth capturing group.

? – Matches the previous group zero or one times.

: - Matches “:” literally.

( - Begin seventh capturing group.

( - Begin eighth capturing group.

\( - Matches “(“ literally.

-?\d+ - Any integer positive or negative including zero.

, – Matches “,” literally.

-?\d+ - Any integer positive or negative including zero.

\) – Matches “)” literally.

) – End eighth capturing group

+ - Match the previous group 1 or more times.

) – End seventh capturing group.

$ - Assert that the match must end at the end of the string.

/ - End the regular expression.

g – Enable the global flag, which means do not stop after the first match (mostly irrelevant here).

Source Code