HOMEWORK #1:

Linear Logistic Classification

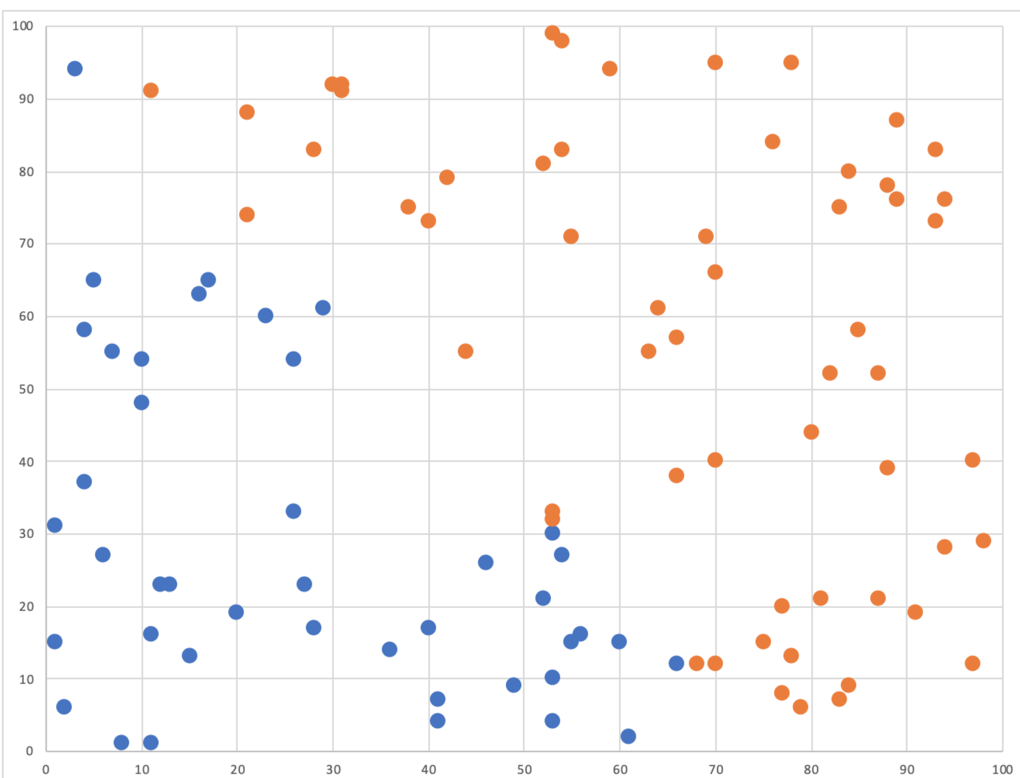
Due Date: Thursday, December 9th, 11:59:59pm

Problem:

Write a single-neuron **Logistic Classifier,** a program that will learn to classify inputs from a collection of examples.

Input:

The file `train.txt` contains the training data for this classification problem. The first two columns are the **inputs** of each example (from 0 to 100). The third column is the classification of the example inputs. (0 or 1). The file `valid.txt` contains the **validation** data for this classification problem. You can visualize the training data in the following plot:



Gradient Descent:

You will implement your Logistic Classifier using **incremental gradient descent**, and will use the Sum-of-Squares as the error measurement for your classifier. Your objective is then to minimize the function:

$$SSE(E) = \sum_{e \text{ in } E} (y(e) - y^{\wedge}(e))^2$$

where E is the set of examples, $y(e)$ is the class value for an example e, and $y^{\wedge}(e)$ is the output of the classifier, given by

$$z = \sum_{i=0}^{n} (w_i * x_i(e))$$
$$y^{\wedge}(e) = sig(z)$$

Where $sig()$ is the sigmoid function given by

$$sig(z) = 1 / (1 + e^{-z})$$

Recall also the derivative of the sigmoid function:

$$sig'(z) = sig(z) * (1 - sig(z))$$

If the error for a single example e is:

$$(y(e) - y^{\wedge}(e))^2$$

Then the *partial derivative* of this error with respect to a single weight w_i is:

$$(\partial \text{Error} / \partial y^{\wedge}(e)) * (\partial y^{\wedge}(e) / \partial z) * (\partial z / \partial w_i)$$
$$=$$
$$-2*(y(e) - y^{\wedge}(e)) * y^{\wedge}(e) * (1 - y^{\wedge}(e)) * x_i(e)$$

The constant 2 is latter absorbed into the learning rate $\boldsymbol{\eta}$ (eta).

Notice that we are applying the sigmoid function as a **squash function**. This is because we are doing *classification*. Also notice that this problem has **two** inputs, which means that the classifier is going to be learning **three** weights.

You shall implement **batch** gradient descent. This means that you will be updating the learner's weights after each batch. Initialize the weights randomly with real numbers **in the range [-2..2]**. The **learning rate $\boldsymbol{\eta}$** (eta), the **number of iterations** and the **batch size** is **left for you to decide**. Experiment with different values and search for one that yields good results.

Validation:

After performing gradient descent on the training data, you should evaluate the performance of your learner against the validation data by computing the sum-of-squares error between the validation data and your learner's predictions. Do **not** use the validation data to train!.

Submission Guidelines:

You will submit through **Canvas**:
Your submission should consists of the following components:

1. Your **program** files.- Submit all necessary files. Your main program file should be called '`learner1.`*X*' where X is the extension of whatever programming language you are using.
2. A **text** file, called '`learner1output.txt`', in which you report on the training run that achieved the best results. This file should **strictly** follow the format shown below.

Report file format:

The first line of the report text file is the values of the learned weights, space separated. The second line is the **Sum of Squares** error of the learned weights against the validation data. It is important that you follow the format because **these lines will be read by the auto-grader** to evaluate your program.

Sample '`learner1output.txt`'

```
3.1415 42.0 1.0
1237491.2831

CS-5001: HW#1
Programmer: Philip J. Fry

TRAINING:
Using learning rate eta = 0.001
Using 25000 iterations.

LEARNED:
w0 = 3.1415
w1 = 42.0
w2 = 1.0
```

```
VALIDATION
Sum-of-Squares Error = 1237491.2831
```

Pseudocode:

PROCEDURE Logistic_Regression_Learner
GIVEN:
    Ex[0..n] : examples, each a $\langle X_0 .. X_i, Y \rangle$ tuple.
        (remember, in every example, $X_0 = 1$)
LOCAL:
    w[0..i] : weights.
    Randomize w[0..i].

    REPEAT no_iterations
        FOR EACH example e = $\langle X_0 .. X_i, Y \rangle$ DO
            compute yCap (from w[0..i] and $X_0 .. X_i$)
            delta := (Y - YCap) * YCap * (1 - Ycap)
            FOR EACH weight w[k] DO
                update w[k] by Eta * delta * X_k

    OUTPUT w[0..i]r3

END.