Exercise #1 : Training One Neuron to Classify the type of Iris.

Prepared by Dr. Alex Frid

Submission Date: Dec 26 (*before* class)

Any delay will result in a fifteen point penalty the first day; and ten points for each subsequent week. (So submit on time.) You can and should work in teams of two; but I will accept work from individuals or triples. All members of the team must meet me together. The submissions will be a physical report (i.e. not electronic) and runnable code.

1 Classification

In *supervised learning* a *training dataset* $\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$ is given, and the task is to find a function $y = f(x)$ such that the function is applicable to unknown patterns x. Based on the nature of y, we can classify those tasks into two classes:

Classification is to assign a predefined label to an unknown pattern. For instance, given a picture of an animal, we need to identify if that picture is of a cat, or a dog, or a mouse, etc. If there are two categories (or two classes), the problem is called binary classification; otherwise, it is multi-class classification. For the binary classification problem, there is a special case, where patterns of the two classes are perfectly separated by a hyper-plane (see Figure 1). We call the phenomenon *linear separability*, and the hyper-plane *decision boundary*.

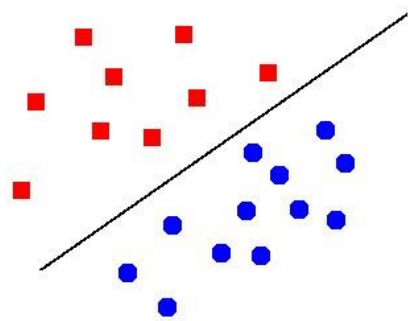

Figure 1: Example of linear separability.

2 Perceptron

A perceptron is a simplified neuron receiving inputs as a vector of real numbers, and outputing a real number (see Figure 2). Mathematically, a perceptron is represented by the following equation

$$y = f(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b) = f(\mathbf{w}^T\mathbf{x} + b)$$

where $w_1,\ldots,w_n$ are weights, b is a bias, $x_1,\ldots,x_n$ are inputs, y is an output, and f is an activation function. In this section, we will use the threshold binary function (see Figure 3)

$$f(z) = \begin{cases} 1 & if\ z > 0 \\ 0 & otherwise \end{cases}$$

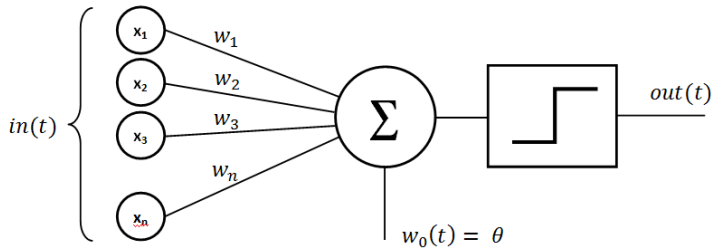

Figure 2: Perceptron

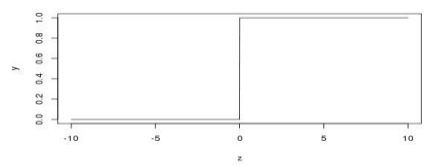

Figure 3: Threshold binary activation function.

2.1 Prediction

A new pattern $\mathbf{x}$ will be assigned the label $\hat{y} = f(\mathbf{w}^T\mathbf{x}+b)$.

The mean square error (MSE) and classification accuracy on a sample $\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)\}$ are respectively defined as

$$error = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2\,, accuracy = \frac{1}{m}\sum_{i=1}^{m} I_{y_i}(\hat{y}_i)^2$$

where $I_u(v)$ is an identity function, which returns 1 if $u = v$ and 0 otherwise.

2.2 Training

Traditionally, a perceptron is trained in an online-learning manner with the delta rule: we randomly pick an example $(\mathbf{x},y)$ and update the weights and bias as follows

$$\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + \eta(y - \hat{y}_{old})\mathbf{x} \quad (1)$$

$$b_{new} \leftarrow b_{old} + \eta(y - \hat{y}_{old}) \quad (2)$$

where η is a learning rate ($0 < \eta < 1$), $\hat{y}_{old}$ is the prediction based on the old weights and bias. Intuitively, we only update the weights and bias if our prediction $\hat{y}_{old}$ is different from the true label y, and the amount of update is (negatively, in the case $y = 0$) proportional to $\mathbf{x}$.

To see why it could work, let apply the weights and bias after being updated to that example where we will assume for the moment that the old classification was 0 but the correct one is 1.

We know $\mathbf{w}_{new}^T\mathbf{x} + b_{new} = (\mathbf{w}_{old} + \eta(y - \hat{y}_{old})\mathbf{x})^T\mathbf{x} + b_{old} + \eta(y - \hat{y}_{old})$

$$= \mathbf{w}_{old}^T\mathbf{x} + b_{old} + \eta(y - \hat{y}_{old})\,(\mathbf{x}^T\mathbf{x} + 1) > \mathbf{w}_{old}^T\mathbf{x} + b_{old}.$$

Since we assumed that $\hat{y}_{old} = 0$, then $\mathbf{w}_{old}^T\mathbf{x} + b_{old} < 0$.

We just showed that $\mathbf{w}_{new}^T\mathbf{x} + b_{new} > \mathbf{w}_{old}^T\mathbf{x} + b_{old}$.

In other words, the update is to pull the decision boundary to a direction that making prediction on the example is 'less' incorrect (see Figure 4).

If the training dataset is linearly separable, it is guaranteed to find a hyperplane correctly separates the training dataset (i.e., *error* = 0) in a finite number of update steps.

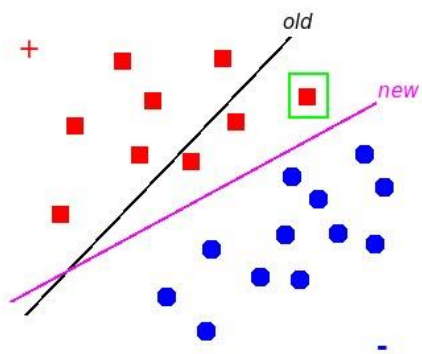

Figure 4: Example of weight update. The chosen example is enclosed in the green box. The update pulls the plane to a direction that making prediction on the example is 'less' incorrect.

3 Exercise

A. Compute by yourself, given the data-points, starting weights are (0,0), the initial threshold is 0, the learning rate is 1:
x1 = (0, 0) y1 = 0
x2 = (0, 1) y2 = 1
x3 = (1, 0) y3 = 1
x4 = (1, 1) y4 = 1

Pattern	Target	Weight Vector	output	*new weight vector*

B. Implement the function *my_perceptron_train*(X, Y), where X is matrix of number of data points by the number of features and Y is a vector (of target values).
The function should train the perceptron on the instances in matrix X.
The function should return 'theta' – the vector with the final parameters and 'k' – the number of updates.

C. Implement function *my_perceptron_test*(theta, X_test, y_test), where:
'theta' – is the classification vector (from B.)
X_test – is matrix of instances that is used for test
Y_test – is vector of target values (labels)
And returns the classification answer

D. Load the Fisher Iris dataset that we looked on in class (also can be downloaded from here http://archive.ics.uci.edu/ml/datasets/Iris) into the memory, select two out of the three groups of Irises and two out of four features (in order to be able to visualize the results). You should play around with the choice of groups of Irises to see if you can tell which sets are linearly separable.
Divide the data randomly into the train and test sets and use the *my_perceptron_train* function to train the perceptron on the training data.
Verify that there is no classification mistakes on the training data.
What is the angle between the 'theta' and the vector (1,0)?
How many steps ('k') needed for the algorithm to converge? Does different divisions into train/test lead to different 'k'?

E. Run the classification on the test set using *my_perceptron_test* function.
Report the classification results you got.
Play with different % of division into train/test groups and report what you see.

Note:
In your implementation you should use visualization of the dataset which is easy to do if you've chosen only two features (i.e. just graph the points in the plane and indicate by color or square/circle which class the point belongs to. Also visualize the learning process (i.e. visualize the separation plane in each iteration).
You should use these visualizations in your final report.