The Perceptron

Mgr. Michal Odler

22.6.2022

What si the Perceptron?

Perceptron was first implemented by McCulloch and Pitts in 1943 and serves as the formal mathematical definition of an *artificial neuron*, similar to those used in artificial neural networks. It is a binary classifier, meaning it is capable of classifying binary data (observations that belong to one of two classes, usually denoted negative and positive class). The perceptron contains weights w_i corresponding to various linear predictiors (data points, denoted as x_i) and a bias b. These values are multiplied and summed together, giving us a *pre-activation value*. This value serves as an input for the activaiton function, which is in this case the Heaviside function. The perceptron's output is then determined as follows:

$$f(x) = 1, \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i + b \geq 0, f(x) = 0, \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i + b < 0.$$

If we inspect the pre-activation value $\sum_{i=1}^{n} w_i x_i + b$ from a geometric point of view, we see that the boundary $\sum_{i=1}^{n} w_i x_i + b = 0$ defines a hyperplane in the feature space. This leads to a major limitation of the perceptron; it can only classify linearly separable data, which are very rarely encountered in the real world. This limitation is dealt with by connecting many perceptrons together with different choices of nonlinear activation functions, effectively creating algorithms we call feed-forward neural networks (also *multi-layer perceptrons*).

Despite this limitation, the perceptron serves as a great educational tool as it shows the inner workings of artificial neurons, and therefore the information flow in neural nets.

How does the percetpron learn from data?

In the beginning, the weights and bias of percepotron are usually chosen as small random numbers. The parameters are then updated by a heuristic rule, which can be shown to converge to a hyperplane that separates the observed data. Most sources present the following algorithm:

$$w_i := \begin{cases} w_i & y_j = \hat{y}_j \\ w_i - \eta x_{i,j} & y_j = 0 \text{ and } \hat{y}_j = 1 \\ w_i + \eta x_{i,j} & y_j = 1 \text{ and } \hat{y}_j = 0 \end{cases}$$

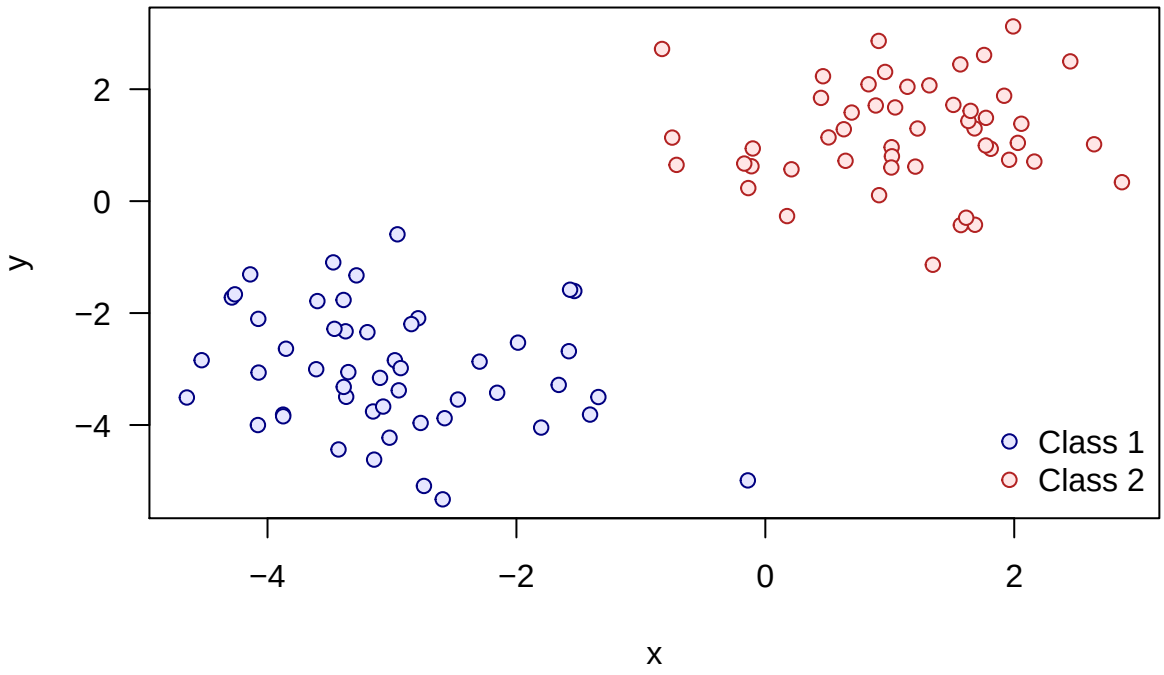
We see that the rule updates the weights for those observations, which have been misclassified. The individual weights are updated by subtracting/adding the value of corresponding predictor, depending on what type of misclassification we are dealing with. The size of the updates can be controlled by the hyperparameter η, also called the *learning rate*.

This rule of learning might seem unintuitive, but it can be easily derived from the *gradient descent* algorithm, used for minimalization of the loss function in neural networks. There needs to be some technical work done, e.g. the change of loss function, as the derivative of Heaviside function is zero *almost everywhere*.

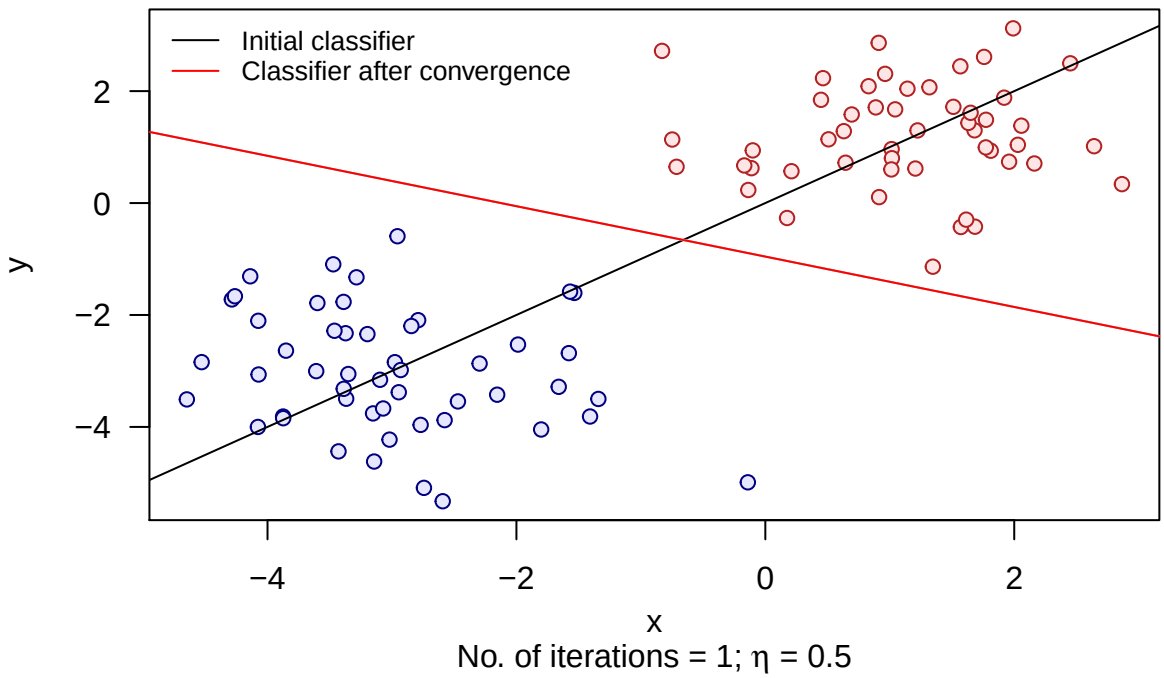
The learning algorithm can be cycled throughout the observed data until convergence. However, as mentioned previously, the algorithm cannot converge if the data is not linearly separable.

Implementation

Let us look at our implementation of the perceptron. For this showcase, I generated two clouds of data from two bivariate normal distributions.

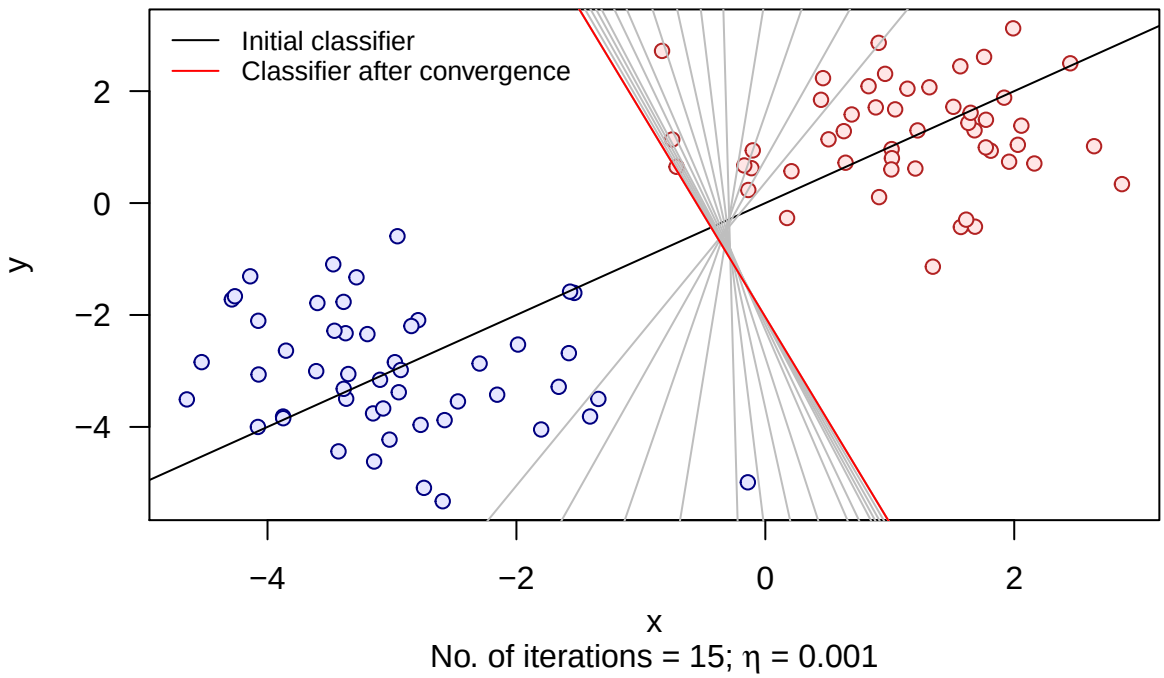


The data is linearly separable, so the perceptron should have no troble converging to a hyperplane that separates the two data clouds. The learning rate has been set to $\eta = 0.5$. The intial weights have been set to $\mathbf{w} = (0, 0.1, -0.1)$, which correspond to a simple $y = x$ line.



[1] 0.6572774 0.3093234 0.6872448

As we can see, the perceptron only needed one iteration to find a hyperplane (a line in our 2D case) that separates our data. If we want to see the individual corrections the learning algorithm is making, we need to set a smaller learning rate. Let us try $\eta = 0.001$:



[1] 0.4711727 0.8511583 0.2313564

In the second case, the algorithm needed way more iterations to find the separating hyperplane. It is also worth pointing out that the separating line found with very low learning rate is very close to some of the 'boundary' data points, which means it is far from optimal as the perceptron would have trouble predicting the category of new observations. Sadly, we cannot control quality of the separator; for this we would need more elaborate ML algorithms. The so called *support vector machines* are designed to find not only *some* separator, but to ensure some degree of optimality.

Higher dimensions

In the previous case, we demonstrated the funcionality of perceptron on a simple dataset, with only two predictors. However, the algorithm works in any number of dimensions, as long as the data are linearly separable by a hyperplane. The visualisation is problematic with number of predictors higher than 3. I extended the perceptron function for higher dimensions, but since I had some problems with embedding interactive 3D plots into RMarkdown, I will unfortunately not be showing the results.

Conclusion

The perceptron may be a simple algorithm that has almost no use in today's world, but it works as a great educational tool to ilustrate the inner workings of artificial neurons. In my opinion, the perceptron should be the first topic for everyone that is trying to get into machine learning topics that revolve around neural networks. This short project is an initial description and demonstration of its functionality.