Perceptron Algorithm Guide

Xavier Huijts
xavierhuijts@gmail.com

In this guide, I aim to explain the *Perceptron* model in a simple and intuitive way to help readers better grasp the underlying notations and optimization process. To achieve this, I provide a practical example. Additionally, I will provide a Python notebook (modeled from scratch, not using packages) on my GitHub account, allowing you to execute the model using a sample DataFrame of features (also known as the *independent variables*, *predictors*, or *input*) and a target (also known as the *dependent variable*, *response variable*, *output*, or how I like to put it: the actual thing we attempt to predict..).

1 Introduction

The Perceptron, introduced by Frank Rosenblatt in 1958, is one of the earliest machine learning models and serves as a fundamental building block for modern neural networks. It is a binary classifier that maps input features to one of two possible outputs (e.g., 0 or 1, or in more intuitive terms: 'dog' or 'cat'). The Perceptron works by assigning weights to inputs, calculating their weighted sum, and applying an activation function to make a prediction. The weights essentially decide the importance of the feature(s). If the prediction is incorrect, the model adjusts the weights based on the error, repeating this process over many iterations (or several epochs, where the entire dataset is used multiple times) to progressively improve accuracy.

The number of epochs is a key parameter: more iterations allow the model to learn better, but excessive training can lead to over-fitting, where the model becomes too tailored to the training data and struggles to generalize. Finding the right balance in optimization is crucial for achieving a model that is both accurate and robust to new, unseen data.

The Perceptron model differs fundamentally from linear regression (e.g., least squares regression) in its optimization process. Linear regression minimizes a loss function, such as the mean squared error, over the entire training data-set to find optimal parameters. In contrast, the Perceptron model uses a mistake-driven learning approach, updating its parameters iteratively after processing each individual misclassified instance (online learning). This can make the model very useful, for example, for streaming data.

As shown in the figure 1, the data must be linearly separable for the Perceptron model to function. In the left figure, the data is perfectly separable by a single linear boundary (a rare scenario in practice but useful for illustration). In contrast, the right figure shows data that cannot be correctly classified with a single linear boundary. To solve this,an additional Perceptron layer is required, which I will not further cover here.

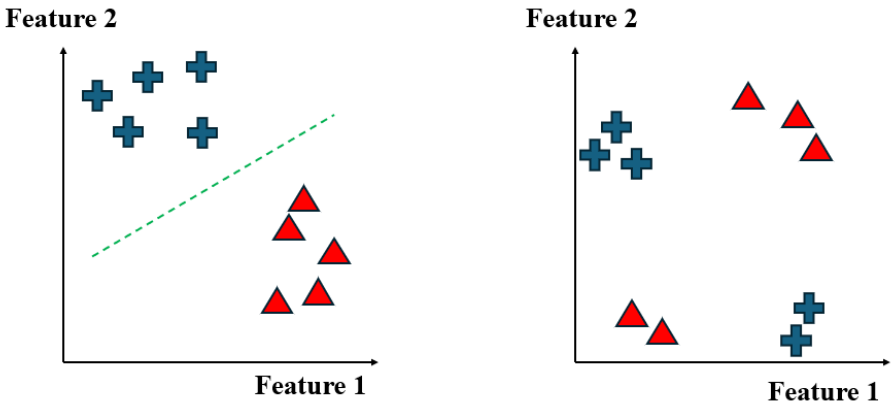

Figure 1: Linearly Separable vs. Non-Separable Data in the One-Layer Perceptron

2 Notations

In this section, I define the key math notations used in the Perceptron model.
In a dataset with only one feature, the instances (also known as *observations* or often *rows*) can be represented as a vector (X) with m instances (e.g., a data-set of student names may contain 100 [m] names):

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix}$$

A feature can be described as a characteristic of the instance, for example, a student's name, age, or home-address. If there are multiple features ($n > 1$), the data-set can be represented as a matrix, where m corresponds to the number of instances and n to the number of features (often *columns* in practice). A matrix of m instances and n features is structured as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_{1,1} & \mathbf{X}_{1,2} & \cdots & \mathbf{X}_{1,n} \\ \mathbf{X}_{2,1} & \mathbf{X}_{2,2} & \cdots & \mathbf{X}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{X}_{m,1} & \mathbf{X}_{m,2} & \cdots & \mathbf{X}_{m,n} \end{bmatrix}$$

In a dataset where the goal is to predict a single value (e.g., a student's GPA) for a given instance i (or student), the target variable can be represented as a vector of Y-values (with m being the total number of instances, with each a corresponding Y value):

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}$$

Each instance i (e.g., a single student in a school's database) is represented by a set of n features (again, which can be the student's name, age, or home-address), forming a feature vector $\mathbf{X}_i$, and an associated target value $\mathbf{Y}_i$ (which is the value we aim to predict). The feature vector $\mathbf{X}_i$ for instance i is structured as (with n being the number of features):

$$\mathbf{X}_i = [\mathbf{X}_{i,1}, \mathbf{X}_{i,2}, \ldots, \mathbf{X}_{i,n}]$$

Since the Perceptron algorithm is focused on binary outcomes, the target value $\mathbf{Y}_i$ is restricted to one of two possible values. For this example, I have selected the values -1 and 1, although this is an arbitrary choice. To make this more intuitive, we could perceive -1 as a *cat* and 1 as a *dog* (which have to be quantified for optimization purposes):

$$Y_i \in \{-1, 1\}$$

We aim to predict the outcome Y_i for instance i, using its corresponding features ($\mathbf{X_i}$) by learning a weight vector W and bias term (b):

$$W = [W_1, W_2, \ldots, W_n].$$

The weight vector (W) and bias (b) are updated iteratively, one instance at a time, during each iteration. It is important to note that the size of W must match the number of features n. Initially, W and b are typically initialized to random values, for example, with zero or one as common starting points.

3 Model Formulation

The Perceptron computes a weighted sum of n features for observation i and adds the bias term (b) for a given instance i:

$$f(\mathbf{X_i}) = W_1\mathbf{X_{i,1}} + W_2\mathbf{X_{i,2}} + \cdots + W_n\mathbf{X_{i,n}} + b$$

The goal of the Perceptron algorithm is to learn the weight vector W and bias term b such that the predicted output $\hat{Y}_i$ matches the true label Y_i for all training instances. To make a prediction ($\hat{Y}_i$) we apply a non-linear

activation function to the weighted sum. The Perceptron uses a threshold-based activation function (making it non-linear), which outputs:

$$\hat{Y}_i = \begin{cases} 1, & \text{if } f(\mathbf{X_i}) \geq 0, \\ -1, & \text{if } f(\mathbf{X_i}) < 0. \end{cases}$$

or in a more simplicity notation:

$$\hat{Y}_i = sign(f(\mathbf{X_i}))$$

This activation function is known as the $signfunction$ and determines the predicted target based on the 'sign' (whether it is positive or negative) of the weighted sum plus the bias.

4 Optimization Process

The Perceptron model updates the weights W and bias b iteratively to minimize classification error. The parameters W and b are only updated in case of a prediction error, meaning the predicted $\hat{Y}$ for a given instance i does not match the true label Y ($\hat{Y}_i \neq Y_i$). The update rules are as follows:

$$W \leftarrow W + \Delta W$$

$$b \leftarrow b + \eta \cdot Y_i$$

where the weight update ΔW is given by:

$$\Delta W = \eta \cdot Y_i \cdot \mathbf{X_i}$$

Here:

- η: the learning rate, a small positive constant, a scalar.
- Y_i: the true y label (for a given instance i), a scaler.
- $\mathbf{X_i}$: a scaler (for a given instance i) in case $n = 1$, a vector in case $n > 1$.

Since we use a learning rate (η) of 1 we can simply rewrite the update rules for b and Δw as follows:

$$b \leftarrow b + Y_i$$

$$\Delta W = Y_i \cdot \mathbf{X_i}$$

The perceptron optimization algorithm can be summarized as follows (this process is usually repeated until a number of iterations is reached):

Algorithm 1 Perceptron Algorithm for Weight Update (an example with # $epochs = 1$)

```
Initialize weights W and bias b
for each instance i do
    Compute the predicted output Ŷ_i = sign(X_i · W + b)
    if Ŷ_i = Y_i then
        No update needed, continue to the next instance
    else
        Update weights: W ← W + Y_i · X_i
        Update bias: b ← b + Y_i
    end if
end for
```

5 Practical Example

5.1 Instance and Feature Notation

To illustrate, consider two instances, each characterized by two features and a binary outcome. Let i represent the instance index, and j represent the feature index. When $\mathbf{X}$ only has one subscript (as in the example in the below, with $\mathbf{X_1}$ and $\mathbf{X_2}$, I refer to instance i and all its corresponding features. If there are two subscripts ($\mathbf{X_{i,j}}$), I refer to the specific feature j of instance i.

Here is an example dataset with two instances ($\mathbf{X_1}$ and $\mathbf{X_2}$):

$$\mathbf{X_1} = [2, 1], \quad Y_1 = 1$$

$$\mathbf{X_2} = [1, 3], \quad Y_2 = -1$$

For simplicity, the data can be represented in tabular form:

Instance (i)	Feature 1 ($\mathbf{X_{i,1}}$)	Feature 2 ($\mathbf{X_{i,2}}$)	Target (Y_i)
1 ($\mathbf{X_1}$)	2	1	1
2 ($\mathbf{X_2}$)	1	3	-1

5.2 Initialize Weights and Bias Term

Assume the following randomly initialized weights (W) and bias (b):

$$W = [1, 0], \quad b = 0$$

The initial decision function is:

$$f(\mathbf{X_i}) = W_1\mathbf{X_{i,1}} + W_2\mathbf{X_{i,2}} + b = \mathbf{X_{i,1}}$$

By quickly graphing this function and the corresponding two instances in figure 2, we can visualize the problem. It is clear that the first instance ($\mathbf{X_1}$) is correctly predicted (being in the area returning 1) ($\hat{Y}_i = Y_i$). However, the second instance ($\mathbf{X_2}$) is also predicted as 1, while the true (Y_i) value is -1 ($\hat{Y}_i \neq Y_i$).

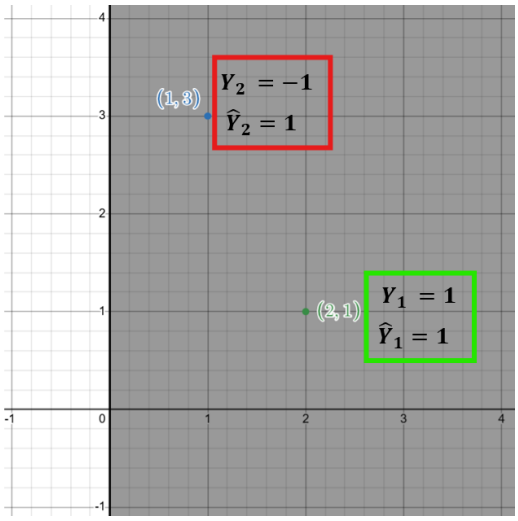

Figure 2: function for $f(\mathbf{X_i}) = \mathbf{X_{i,1}} \geq 0$

5.3 Iteration 1

While the initial visualization of the decision function $f(X_i))$ in Figure 2 provides insight into its performance, the function fails to correctly classify both instances in the dataset. To address this, we now begin the optimization process to improve the decision boundary. We start with the first instance, please recall:

$$\mathbf{X_1} = [2, 1], \quad Y_1 = 1$$

Using the initialized weights $W = [1, 0]$ and bias $b = 0$, we compute the weighted sum:

$$f(X_1) = W_1\mathbf{X_{1,1}} + W_2\mathbf{X_{1,2}} + b = 2 \cdot 1 + 1 \cdot 0 + 0 = 2$$

The activation function (sign function) outputs:

$$\hat{Y_1} = sign(2) = 1$$

Since the predicted output $\hat{Y_i} = 1$ matches the true label $Y_i = 1$ ($\hat{Y_1} = Y_1$), no update to the weights or bias is required. Therefore, we proceed to the next instance.

5.4 Iteration 2

We now proceed with the second iteration. Consider the second instance in the dataset:

$$\mathbf{X_2} = [1, 3], \quad Y_2 = -1$$

Using the current weights $W = [1, 0]$ and bias $b =$, the weighted sum is:

$$f(\mathbf{X_2}) = W_1\mathbf{X_{2,1}} + W_2\mathbf{X_{2,2}} + b = 1 \cdot 1 + 3 \cdot 0 + 0 = 1$$

The activation function outputs:

$$\hat{Y_2} = \text{sign}(1) = 1$$

However, the true output $Y_2 = -1$ does not match the predicted output $\hat{Y_2} = 1$ ($\hat{Y_2} \neq Y_2$). We update the weights and bias as follows:

$$\Delta W = Y_2 \cdot \mathbf{X_2} = -1 \cdot [1, 3] = [-1, -3]$$

$$\Delta b = Y_2 = -1$$

The updated weights and bias are:

$$W \leftarrow W + \Delta W = [1, 0] + [-1, -3] = [0, -3]$$

$$b \leftarrow b + \Delta b = 0 + (-1) = -1$$

The new decision function is:

$$f(X_i) = 0\mathbf{X_{i,1}} + -3\mathbf{X_{i,2}} - 1 = -3\mathbf{X_{i,2}} - 1$$

By graphing this new function in figure 3, the second instance is now correctly classified, but the first instance is misclassified. We continue the process.

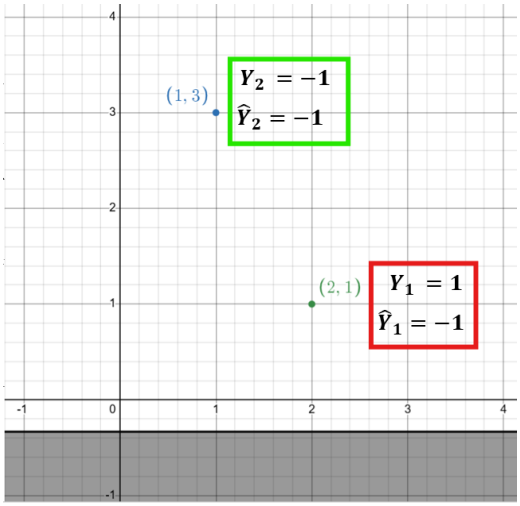

Figure 3: Iteration 2: updated function for $f(\mathbf{X_i}) = -3\mathbf{X_{i,2}} - 1 \geq 0$.

5.5 Iteration 3

In the 3rd iteration, we return to the first instance:

$$\mathbf{X_1} = [2, 1], \quad Y_1 = 1$$

Using the current weights $W = [0, -3]$ and bias $b = -1$, the weighted sum is:

$$f(\mathbf{X_1}) = W_1\mathbf{X_{i,1}} + W_2\mathbf{X_{i,2}} + b = 2 \cdot 0 + 1 \cdot -3 + -1 = -4$$

The activation function outputs:

$$\hat{Y_1} = sign(-4) = -1$$

However, the true output ($Y_1 = 1$) does not match the predicted output ($\hat{Y_1}$) of -1 ($\hat{Y_1} \neq Y_1$). We update the weights and bias as follows:

$$\Delta W = Y_1 \cdot \mathbf{X_1} = 1 \cdot [2, 1] = [2, 1]$$

$$\Delta b = Y_1 = 1$$

The updated weights and bias are:

$$W \leftarrow W + \Delta W = [0, -3] + [2, 1] = [2, -2]$$

$$b \leftarrow b + \Delta b = -1 + 1 = 0$$

The new decision function is:

$$f(\mathbf{X_i}) = 2\mathbf{X_{i,1}} - 2\mathbf{X_{i,2}}$$

By visualizing this updated function in figure 4, we observe convergence toward a decision boundary that correctly classifies both instances. This would also stand while plugging in the numbers, so we are done!

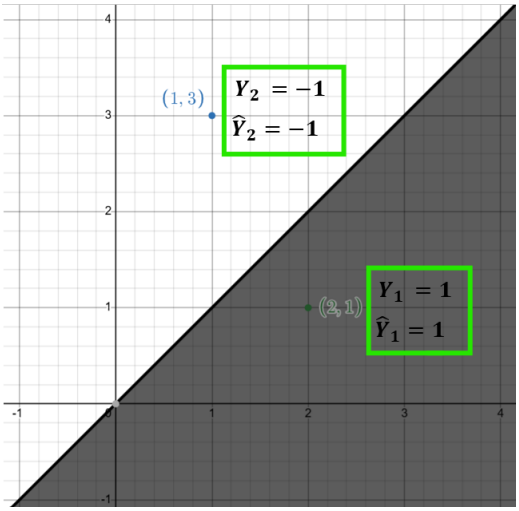

Figure 4: Iteration 3: updated function for $f(\mathbf{X_i}) = 2\mathbf{X_{i,1}} - 2\mathbf{X_{i,2}} \geq 0$

5.6 Final result

As visible in figure 4, the two instances are correctly classified. However, lets now properly prove to by inputting the features into the updated function $f(X_i)$. Let's recall $f(X_i)$:

$$f(\mathbf{X_i}) = 2\mathbf{X_{i,1}} - 2\mathbf{X_{i,2}}$$

Instance one being the following:

$$\mathbf{X_1} = [2, 1], \quad Y_1 = 1$$

By imputing this, we get to following:

$$f(\mathbf{X_1}) = 2 * 2 - 2 * 1 = 2$$

The activation function outputs:

$$\hat{Y_1} = sign(2) = 1$$

Now, the true Y_1 value matches the prediction $\hat{Y_1}$ ($\hat{Y_1} = Y_1$). Let's now verify whether this also counts for the second instance.

Instance two being the following:

$$\mathbf{X_2} = [1, 3], \quad Y_2 = -1$$

By imputing this, we get to following:

$$f(\mathbf{X_2}) = 2 * 1 - 2 * 3 = -4$$

The activation function outputs:

$$\hat{Y_1} = sign(-4) = -1$$

Now, likewise for the second instance, the true Y_2 value matches the prediction $\hat{Y_2}$ ($\hat{Y_2} = Y_2$).

The weight vector W and the bias term b are now correctly optimized for our data! In practice, however, data is rarely perfectly separable. Therefore, the number of iterations is usually predefined rather than continuing until all instances are correctly classified.