3. PERCEPTRON

Consider the problem in supervised learning. In this case, the data is given as a set of pairs

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$$

$x^{(i)}$ = input = independent variable = predictor = attribute
$y^{(i)}$ = output = dependent variable = response = target

Goal: we want to learn the underlying relationship (hypothesis) between x and y, so that next time we are given an instance of an input x, we can predict its corresponding output y.

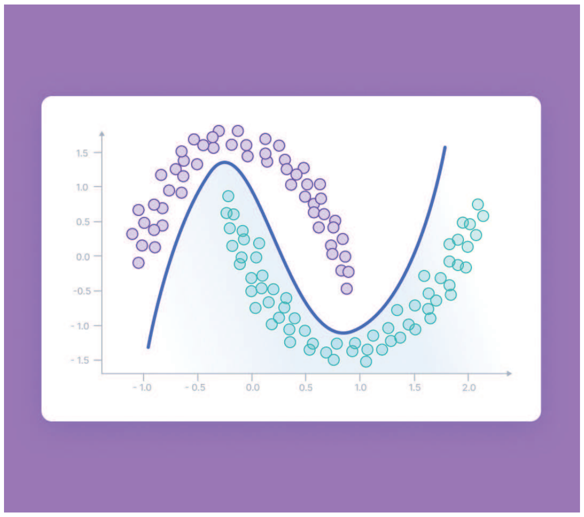
- Typically, $x \in \mathbb{R}^d$ is a vector of d components

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_d \end{bmatrix}$$

  More precisely, x might be a customer, patient, image, text, song, etc., and we need to represent it numerically

$$\varphi(x) = \text{ feature representation } \in \mathbb{R}^d$$

- In this section, we will study a *binary classification* problem where we are trying to separate two classes of data instances (data points, examples, samples). We assume $y \in \{+1, -1\}$, denoting the labels for data instances.



https://www.v7labs.com/blog/semi-supervised-learning-guide

- To create a learning algorithm, we need a hypothesis class $\mathcal{H}$ which is a set of all possible functions (hypotheses) $x \mapsto y$. More precisely,

  $$y = h(x; \theta)$$

  where θ denotes parameters.

  $$x \rightarrow \boxed{h} \rightarrow y$$

  The goal is to find h that agrees with the given data in D and we hope that h will perform well in the future.

- Next, we define what makes one hypothesis better than another. We define the loss (cost, error) function for a single data instance $(x^{(i)}, y^{(i)})$ that measures the error of predicting the class label $\hat{y}^{(i)} := h(x^{(i)})$ when the actual true class label is $y^{(i)}$. We denote the loss by

  $$L(\hat{y}^{(i)}, y^{(i)})$$

  where

  $$\begin{aligned} \hat{y}^{(i)} &\in \{+1, -1\} \quad \text{predicted label} \\ y^{(i)} &\in \{+1, -1\} \quad \text{actual true label} \end{aligned}$$

  Objective: have a small loss on the <u>*new*</u> data

  Proxy: have a small loss on the given training data $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$

  We minimize the loss on the training data

  $$\mathcal{E}_{train}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x^{(i)}), y^{(i)}) \qquad \text{training set error}$$

  and we hope that the loss on the test data (additional data that the algorithm has not seen) $\{(x^{(n+1)}, y^{(n+1)}), (x^{(n+2)}, y^{(n+2)}), \ldots, (x^{(n+n')}, y^{(n+n')})\}$ will also be small

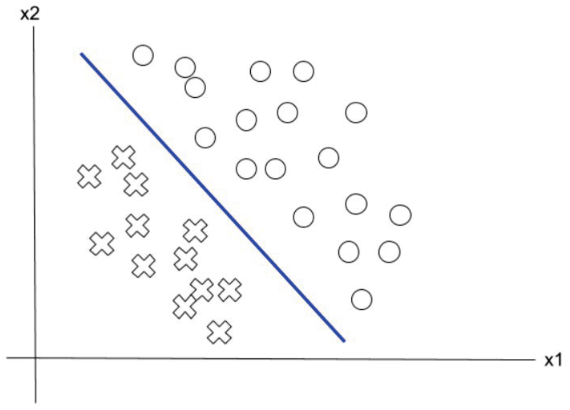
  $$\mathcal{E}_{test}(h) = \frac{1}{n'} \sum_{i=n+1}^{n+n'} L(h(x^{(i)}), y^{(i)}) \qquad \text{test error}$$

- How do we come up with a learning algorithm?

  $$D \rightarrow \boxed{\text{Learning Algorithm}(\mathcal{H})} \rightarrow h$$

  - be a clever human
  - use optimization

- In this section, we consider *linear classifiers*, i.e., we assume that $\mathcal{H}$ is a class of all possible linear separators.



https://automaticaddison.com/linear-separability-and-the-xor-problem/

In particular, we consider a hypothesis of the form

$$\boxed{h(x; w, b) = \text{sign}(w^T x + b) = \begin{cases} +1, & w^T x + b > 0 \\ -1, & w^T x + b < 0 \end{cases}}$$

where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

Recall

$$w^T x = [w_1 \ w_2 \ \ldots \ w_d] \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_d \end{bmatrix} = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \sum_{j=1}^{d} w_j x_j$$

The values of vector w are called weights and b is called a bias.

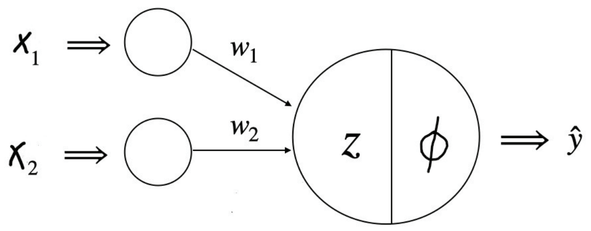
- The algorithm computes the weighted sum of the inputs and if this weighted sum exceeds some threshold (specifically, $-b$), then the predicted label is positive, and if the weighted sum is less than this threshold, the predicted label is negative.

- For simplicity, from now on, we consider the case $d = 2$ with two inputs $(x_1, x_2) \in \mathbb{R}^2$. We can also visualize the above hypothesis as a single neuron.

Given an input (x_1, x_2), the predicted output $\hat{y}$ is found using two steps:

1. the quantity z (pre-activation) is calculated by

$$z = w_1 x_1 + w_2 x_2 + b = \sum_{j=1}^{2} w_j x_j + b$$

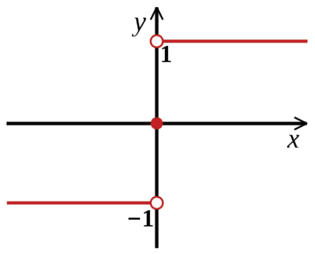


2. the predicted output $\hat{y}$ is calculated by applying the activation function ϕ as

$$\hat{y} = \phi(z).$$

In this case, the activation function is the "sign" function

$$\phi(z) = \begin{cases} +1, & z > 0 \\ 0, & z = 0, \\ -1, & z < 0 \end{cases}$$



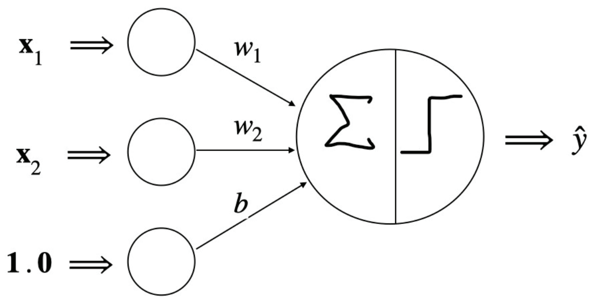
https://en.wikipedia.org/wiki/Sign_function

- To simplify the notation we define

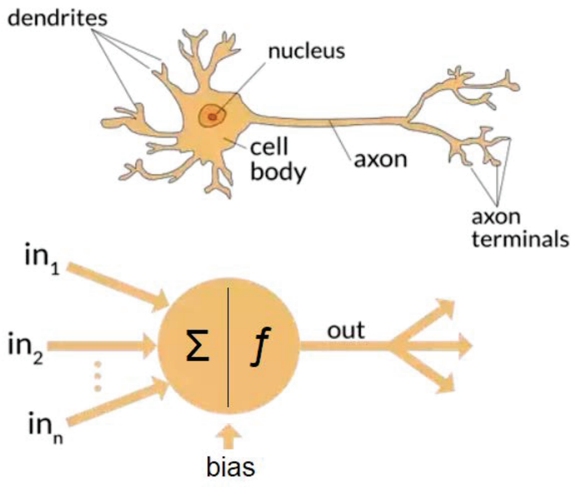
$$\begin{aligned} \bar{x} &= (x_1, x_2, 1) \\ \bar{w} &= (w_1, w_2, b) \end{aligned}$$

Notice that then

$$\begin{aligned} z &= w^T x + b \\ &= w_1 x_1 + w_2 x_2 + b \\ &= \bar{w}^T \bar{x} \\ &= \sum_{i=1}^{3} w_i x_i \end{aligned}$$



- A single artificial neuron is a model of a biological neuron and is a building block for artificial neural networks.



https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

- Note that the separator (decision boundary) is of the form

$$w_1x_1 + w_2x_2 + b = 0$$
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

  which is the equation of the line in the (x_1, x_2)-plane.

- Perceptron

  Given the data set
$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$$
  to find the linear classifier we need to find the model parameters, i.e., the weights w_1, w_2 and the bias b. Perceptron is one of the methods for finding those parameters.

```
perceptron (D, T, $\eta$)
$\bar{w} = \bar{0}$                         initialize the parameter $\bar{w} = (w_1, w_2, b)$ to be the zero vector
for t = 1 to T
      for i = 1 to n
            $\hat{y}^{(i)} = \phi(\bar{w}^T \bar{x}^{(i)})$
            $\bar{w} := \bar{w} - \eta(\hat{y}^{(i)} - y^{(i)})\bar{x}^{(i)}$
return  $\bar{w}$
```

- Here, T is the number of iterations and η is the learning rate of the ML model (typically, a number between 0 and 1). Both T and η are hyperparameters.
- Let us try to understand the update rule for $\bar{w}$ when we are at a data instance $(x^{(i)}, y^{(i)})$

$$\boxed{\bar{w}_{new} = \bar{w}_{old} - \eta(\hat{y}^{(i)} - y^{(i)})\bar{x}^{(i)}}$$

  Recall that $y^{(i)}$ is the actual true class label, while $\hat{y}^{(i)} := h(x^{(i)})$ is the predicted class label computed by our model.

  * If our algorithm did not make a mistake, then $y^{(i)} = \hat{y}^{(i)}$. In that case, $\bar{w}_{new} = \bar{w}_{old}$.
  * If our algorithm misclassified $x^{(i)}$, there are two possibilities:

    Case 1: $y^{(i)} = 1$ and $\hat{y}^{(i)} = -1$
    This means that $x^{(i)}$ is a positive example and it is classified as negative. In other words, this means that the algorithm claims

$$\bar{w}^T \bar{x}^{(i)} < 0,$$

    while actually $\bar{w}^T \bar{x}^{(i)} > 0$. The rule says to modify $\bar{w}$ so that

$$\bar{w} = \bar{w} + 2\eta \bar{x}^{(i)}.$$

    Note that since our algorithm claims $\bar{w}^T \bar{x}^{(i)} = \|\bar{w}\| \|\bar{x}^{(i)}\| \cos\theta < 0$, it means that the angle between $\bar{w}$ and $\bar{x}^{(i)}$ is more than 90. By modifying $\bar{w}$, we ensure that the angle between updated $\bar{w}$ and $\bar{x}^{(i)}$ is less than 90 as it should be.

    Case 2: $y^{(i)} = -1$ and $\hat{y}^{(i)} = 1$
    This means that $x^{(i)}$ is a negative example and it is classified as positive. This means that the algorithm claims

$$\bar{w}^T \bar{x}^{(i)} > 0,$$

    while actually $\bar{w}^T \bar{x}^{(i)} < 0$. The rule says to modify $\bar{w}$ so that

$$\bar{w} = \bar{w} - 2\eta \bar{x}^{(i)}.$$

    Note that since our algorithm claims $\bar{w}^T \bar{x}^{(i)} = \|\bar{w}\| \|\bar{x}^{(i)}\| \cos\theta > 0$, it means that the angle between $\bar{w}$ and $\bar{x}^{(i)}$ is less than 90. By modifying $\bar{w}$, we ensure that the angle between updated $\bar{w}$ and $\bar{x}^{(i)}$ is more than 90 as it should be.

  - If D is linearly separable, the percentron is guaranteed to converge and to produce a classifier. If D is not linearly separable, the perceptron will not converge (see the proof in [2], pages 18-20).
  - Note that the update rule states

$$\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta(\hat{y}^{(i)} - y^{(i)}) \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ 1 \end{bmatrix}$$

    or, in expanded form,

$$\begin{aligned} w_1 &= w_1 - \eta(\hat{y}^{(i)} - y^{(i)})x_1^{(i)} \\ w_2 &= w_2 - \eta(\hat{y}^{(i)} - y^{(i)})x_2^{(i)} \\ b &= b - \eta(\hat{y}^{(i)} - y^{(i)}) \end{aligned}$$

Python code: Lecture_3_Perceptron.ipynb

Homework 1:

- We mentioned that perceptron converges if the data is linearly separable. Try sklearn perceptron model for versicolor and virginica, with sepal length and petal length. What do you observe?

- We created `My_Perceptron` class for only 2 inputs. Extend this code for 3 inputs. Investigate the iris data set and choose 3 features to classify setosa and versicolor using your code. Notice that you cannot easily plot the decision boundary now since the data is 3-dimensional, but you can still compare the actual and the predicted labels to see how your algorithm is performing.

- Try to generalize `My_Perceptron` code so it could be used for any number of inputs. (Hint: Recall, that for a list `w` we can use `w[-1]` and `w[:-1]` to access the last value in the list and all the values expect the very last value. Also, use `np.dot`, NumPy dot product, to compute the pre-activation value of `z`.)

References and Reading Material:

[1] *Hands-On Machine Learning with Scikit Learn, Keras & TensorFlow*, Geron (pages 279-288)

[2] *MIT notes*:
2-LinearClassifiers.pdf
3-Perceptron.pdf (the proof of Perceptron convergence on pages 18-20 is optional)

https://openlearninglibrary.mit.edu/courses/course-v1:MITx+6.036+1T2019/course/