Quando houverem múltiplas explicações possíveis para o mesmo conjunto de dados, opte pela mais simples – Navalha de Ockham.

O Perceptron

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Vetores de características

Um vetor de características de *n* dimensões descreve determinado objeto. Os vetores são na forma $\boldsymbol{x} = [x_1, x_2, ..., x_n]$.

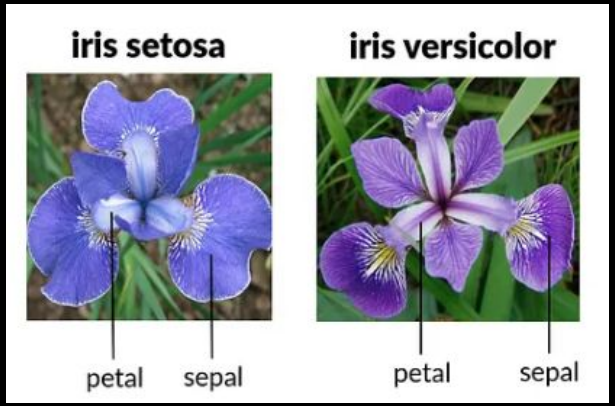
Um vetor de características **descreve determinada instância**.

O vetor de características deve conter dados relevantes para associar a instância a sua **classe alvo** (*target*), que deve ser um valor $y \ni [y_1, y_2, ..., y_k]$, contendo uma das *k* possíveis classes.

Vetores de características

Exemplo: podemos tomar medidas de Comprimento e Largura da Sépala de flores (problema Iris Setosa da aula passada), e usar esses dados como vetores de características.

As flores serão classificadas entre *Iris Setosa* e *Iris Versicolour*. Ou seja, y ∋ [*Iris Setosa*, *Iris Versicolour*].

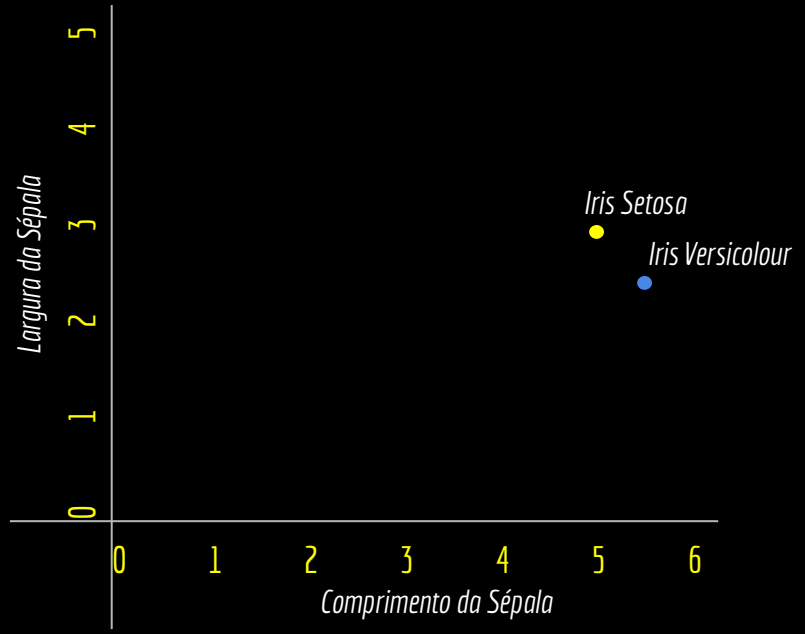


Exemplo

Suponha que um especialista tirou medidas de duas flores, uma *Iris Setosa* e uma *Iris Versicolour*.

x_s = [5,3] y_s = *Iris Setosa*

x_v = [5.5,2.5] y_v = *Iris Versicolour*



Perceptron

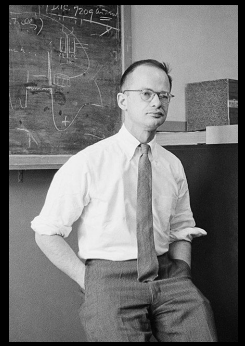
Neurônio artificial capaz de discriminar entre duas **classes**.

Componente básico de uma **rede neural**.

Criado por Frank Rosenblatt.

Inspirado no trabalho de Warren McCulloch e Walter Pitts de 1943.

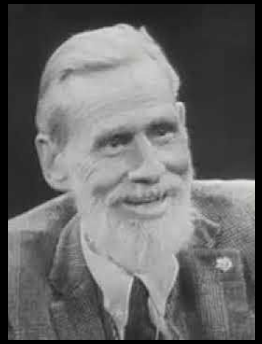
Primeira implementação feita em 1957 em um computador IBM 704.



Walter Pitts
23/04/1923 - 14/05/1969

Lógico Americano.

en.wikipedia.org/wiki/Walter_Pitts



Warren McCulloch
16/11/1898 - 24/10/1969

Neurofisiologista e ciberneticista Americano.

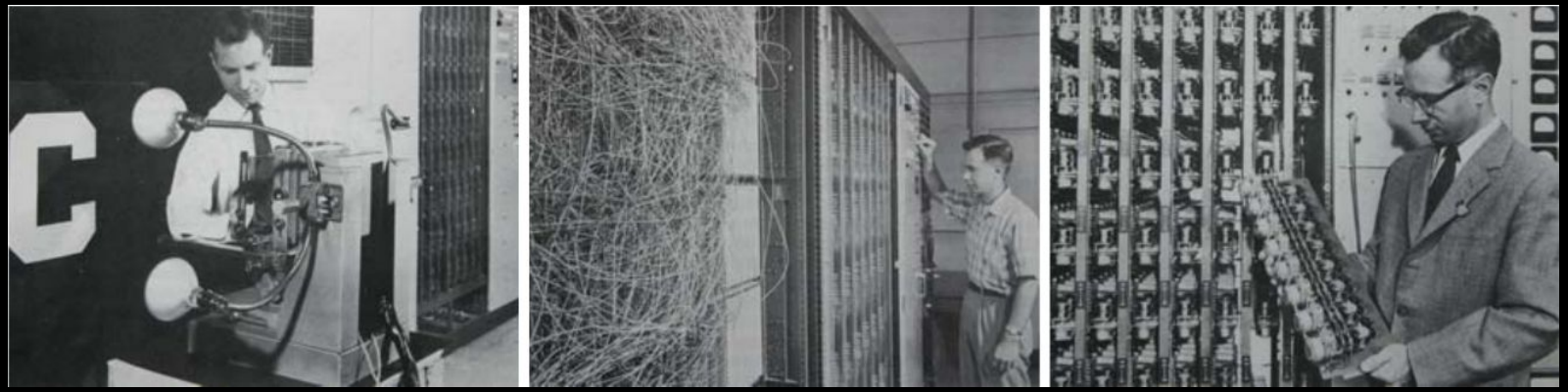
en.wikipedia.org/wiki/Warren_Sturgis_McCulloch

Frank Rosenblatt
11/07/1928 - 11/07/1971

Psicólogo Americano.

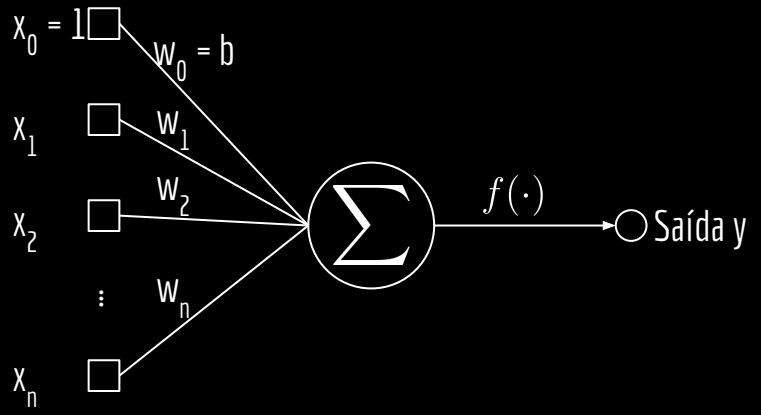
en.wikipedia.org/wiki/Frank_Rosenblatt

Perceptron

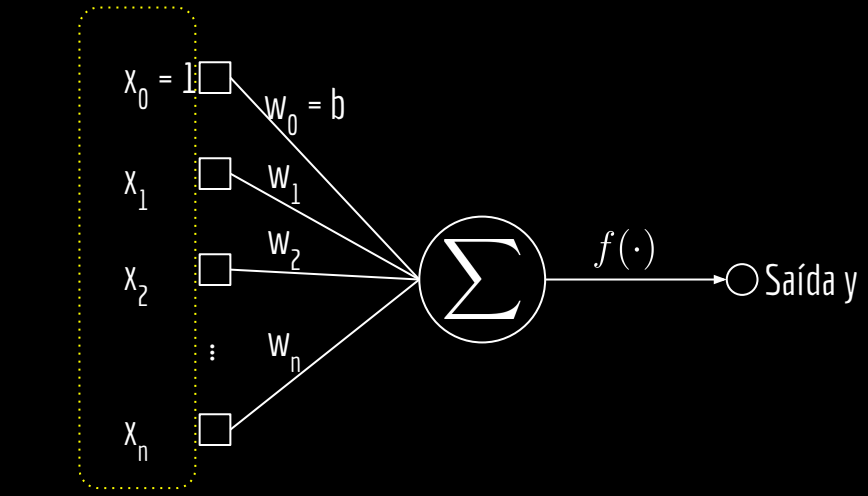


Mark I Perceptron, de 1960.

Perceptron

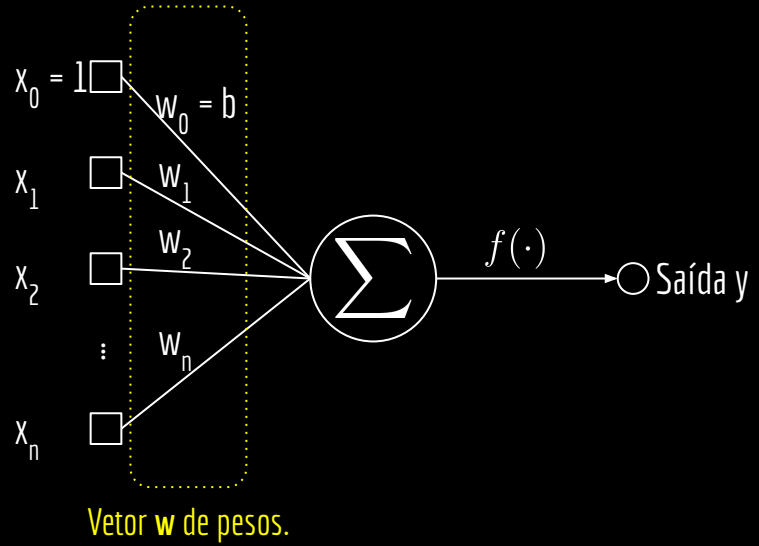


Perceptron

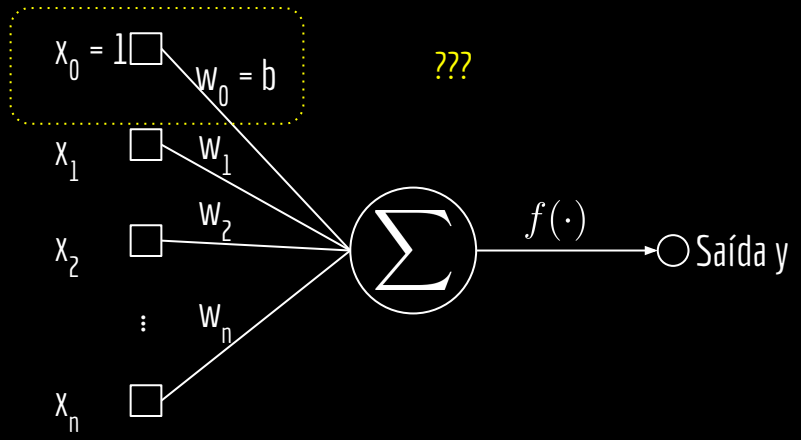


Vetor **x** de entrada.

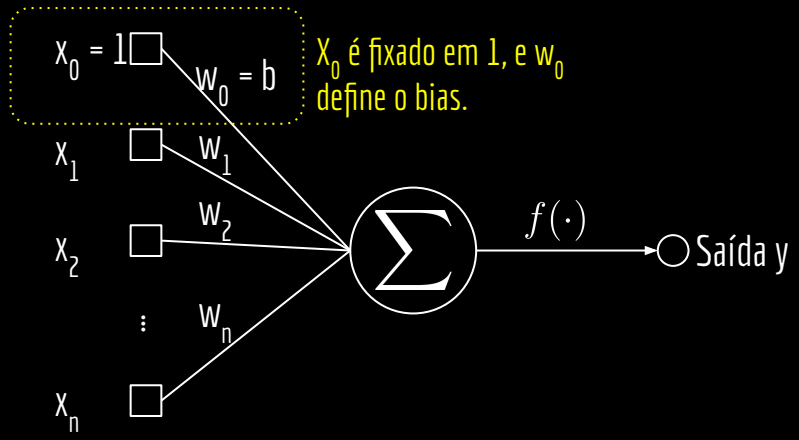
Perceptron



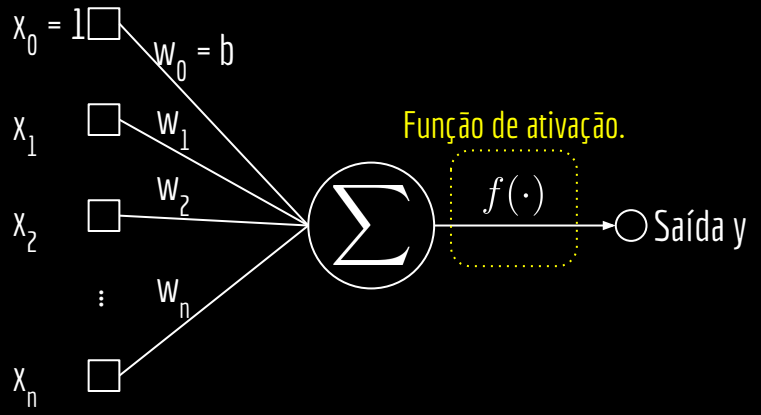
Perceptron



Perceptron


$x_0 = 1$
$w_0 = b$
x_0 é fixado em 1, e w_0 define o bias.
x_1
w_1
x_2
w_2
w_n
x_n
$\sum$
$f(\cdot)$
Saída y

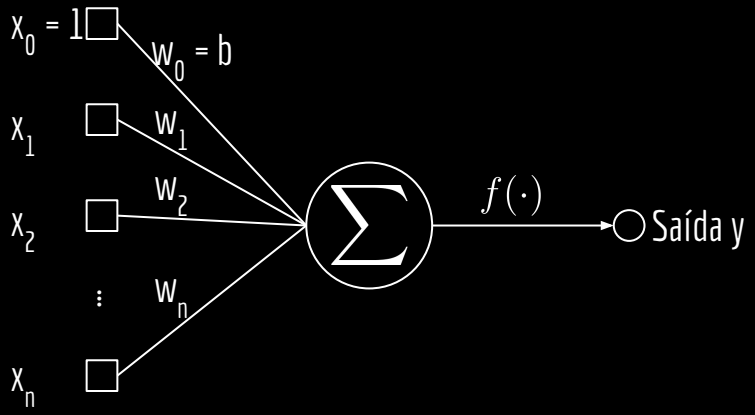
Perceptron



Perceptron

Dado um vetor de características **x** = [x_1, x_2, ..., x_n], o perceptron o classifica em uma de duas classes y ∋ [-1, +1]:

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

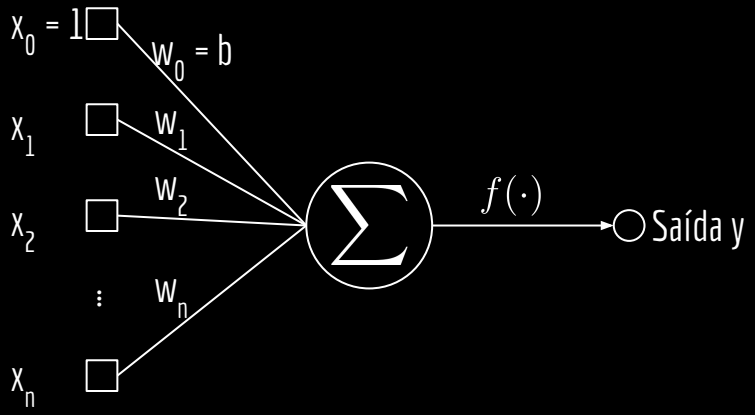


Perceptron

Dado um vetor de características **x** = [x_1, x_2, ..., x_n], o perceptron o classifica em uma de duas classes y ∋ [-1, +1]:

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

???

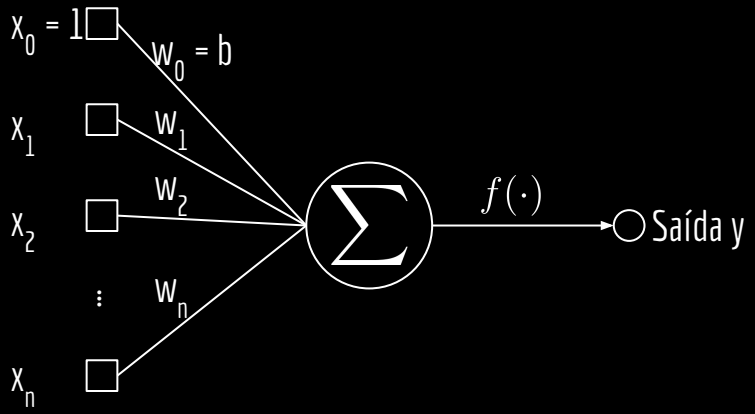


Perceptron

Dado um vetor de características **x** = [x_1, x_2, ..., x_n], o perceptron o classifica em uma de duas classes y ∋ [-1, +1]:

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

Produto escalar.



Função de Ativação

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

A função de ativação $f(\cdot)$ é dada pela função Degrau, definida como:

$$f(a) = \begin{cases} +1, & \text{se } a \geq 0 \\ -1, & \text{se } a < 0 \end{cases}$$

Perceptron

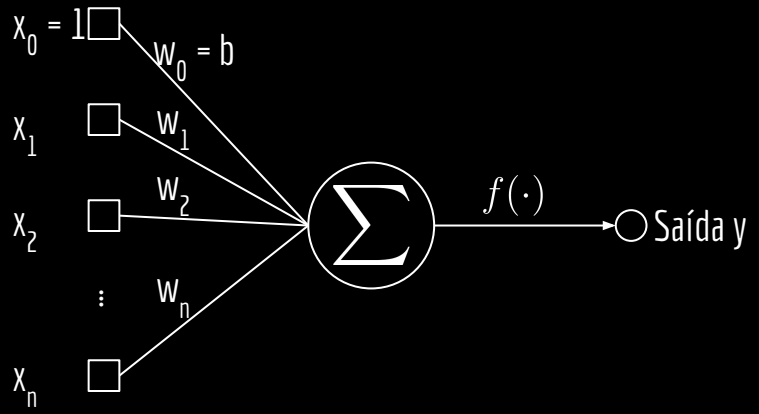
Sendo assim:

Um vetor de pesos **w** define a fronteira de decisão.

O produto escalar entre o vetor de entrada **x** e o vetor de pesos **w** define a classe (positiva ou negativa) de **x**.

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

$$f(a) = \begin{cases} +1, & \text{se } a \geq 0 \\ -1, & \text{se } a < 0 \end{cases}$$



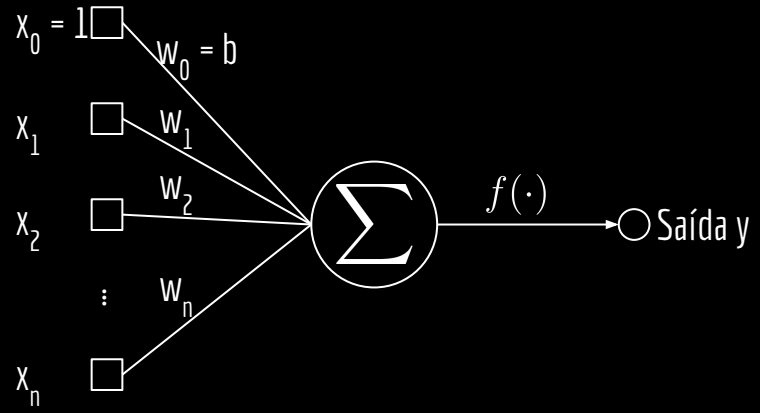
Faça você mesmo #1

Considere os pesos do perceptron disponibilizados no Google Colab.

Use os pesos para classificar os dados entre a classe positiva (Iris Setosa) e negativa (Iris Versicolour).

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x})$$

$$f(a) = \begin{cases} +1, & \text{se } a \geq 0 \\ -1, & \text{se } a < 0 \end{cases}$$



Treinamento

Definir a fronteira decisão de um perceptron de forma automática exigem um algoritmo de **descida de gradiente estocástico**.

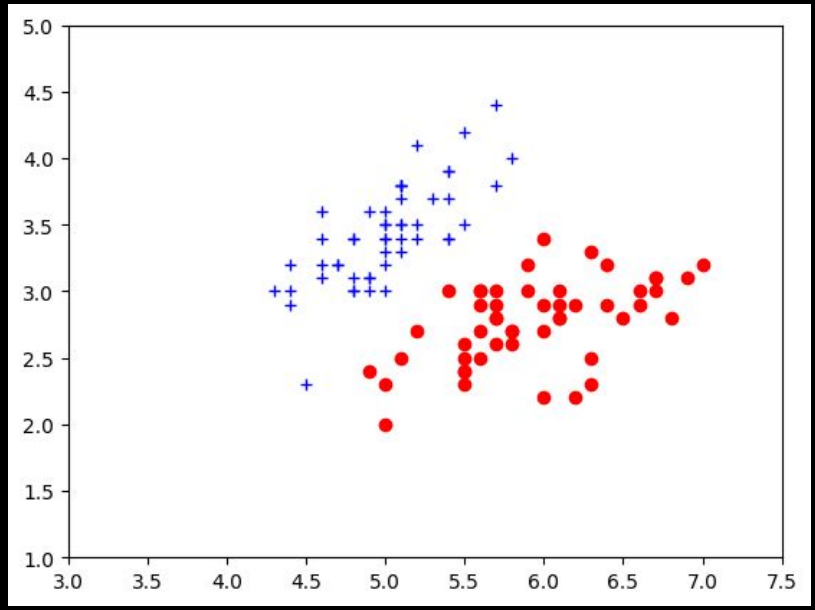
Para isso, são necessários dados de **treinamento rotulados**.

Ideia: aprender com os dados de treinamento para gerar uma fronteira de decisão. Quando novos dados sem rótulo chegarem, essa fronteira será usada para classificação.

Essa é a ideia do **treinamento supervisionado** de um classificador.

Treinamento - Exemplo

Como exemplo, vamos usar todos os dados da versão simplificada do dataset Iris Setosa para treinar um perceptron.



Algoritmo

Considere um dataset de treinamento *M* contendo *n* vetores de características e suas respectivas classes y ∋ [-1, +1].

Desejamos um vetor **w** que satisfaz:

$$f(a) = \begin{cases} \mathbf{w}^T \mathbf{x_i} < 0, & \text{se } y_i \text{ negativo} \\ \mathbf{w}^T \mathbf{x_i} \geq 0, & \text{se } y_i \text{ positivo} \end{cases}$$

Onde x_i e y_i são o vetor de características e a classe da i-ésima instância do dataset *M*.

Algoritmo

Considere um dataset de treinamento *M* contendo *n* vetores de características e suas respectivas classes y ∋ [-1, +1].

Desejamos um vetor **w** que satisfaz:

$$f(a) = \begin{cases} \mathbf{w}^T \mathbf{x_i} < 0, & \text{se } y_i \text{ negativo} \\ \mathbf{w}^T \mathbf{x_i} \geq 0, & \text{se } y_i \text{ positivo} \end{cases}$$

Onde x_i e y_i são o vetor de características e a classe da i-ésima instância do dataset *M*.

Como y_i ∋ [-1, +1], podemos simplificar, buscando um **w** que satisfaz $\mathbf{w}^T \mathbf{x_i} y_i > 0$.

Algoritmo

Inicialize **w** com um valor qualquer.

Passe por cada instância x de treinamento em **M**.
    Se x foi classificada corretamente, não faça nada.
    Se x foi classificada incorretamente, faça:

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Algoritmo

Inicialize **w** com um valor qualquer.

Passe por cada instância x de treinamento em **M**.
    Se x foi classificada corretamente, não faça nada.
    Se x foi classificada incorretamente, faça:

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Ajuste **w** em direção ao vetor **x**.

Algoritmo

Inicialize **w** com um valor qualquer.

Passe por cada instância x de treinamento em **M**.
    Se x foi classificada corretamente, não faça nada.
    Se x foi classificada incorretamente, faça:

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Ajuste **w** em direção ao vetor **x**.
A constante η (eta) define o **fator de aprendizado**, ou *learning rate*.
Qual o tradeoff para valores pequenos ou grandes de η?

Pergunta

Ao ajustar **w** no loop, pode ocorrer de instâncias que eram classificadas corretamente, passarem a ser incorretamente classificadas?

```
Inicialize w com um valor qualquer.

Passe por cada instância x de treinamento em M.
    Se x foi classificada corretamente, não faça nada.
    Se x foi classificada incorretamente, faça:
```

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Algoritmo

Ao ajustar **w** no loop, pode ocorrer de instâncias que eram classificadas corretamente, passarem a ser incorretamente classificadas?

Sim.

Pode ser necessário passar múltiplas vezes pelos dados de treinamento.

```
Inicialize w com um valor qualquer.

Passe por cada instância x de treinamento em M.
    Se x foi classificada corretamente, não faça nada.
    Se x foi classificada incorretamente, faça:
```

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Algoritmo

Inicialize **w** com um valor qualquer.

Enquanto houverem instâncias rotuladas incorretamente:
    Passe por cada instância x de treinamento em **M**.
        Se x foi classificada corretamente, não faça nada.
        Se x foi classificada incorretamente, faça:

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Algoritmo

Inicialize **w** com um valor qualquer.

Enquanto houverem instâncias rotuladas incorretamente:
    Passe por cada instância x de treinamento em **M**.
        Se x foi classificada corretamente, não faça nada.
        Se x foi classificada incorretamente, faça:

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Cada passagem completa pelos dados de treinamento é chamada de **época**.

Pergunta

O algoritmo termina?

Inicialize **w** com um valor qualquer.

Enquanto houverem instâncias rotuladas incorretamente:
    Passe por cada instância x de treinamento em **M**.
        Se x foi classificada corretamente, não faça nada.
        Se x foi classificada incorretamente, faça:
            $\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$

Algoritmo

O algoritmo **nunca converge** se o problema **não** for linearmente separável.

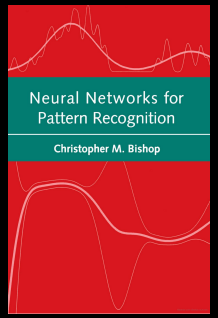
```
Inicialize w com um valor qualquer.

Enquanto houverem instâncias rotuladas incorretamente:
    Passe por cada instância x de treinamento em M.
        Se x foi classificada corretamente, não faça nada.
        Se x foi classificada incorretamente, faça:
```

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Algoritmo

É garantido que o algoritmo converge (termina) se o problema for **linearmente separável**.

Veja prova em Bishop (1995).




```
Inicialize w com um valor qualquer.

Enquanto houverem instâncias rotuladas incorretamente:
    Passe por cada instância x de treinamento em M.
        Se x foi classificada corretamente, não faça nada.
        Se x foi classificada incorretamente, faça:
```

$$\mathbf{w} = \mathbf{w} + \eta(\mathbf{x}y)$$

Faça você mesmo #2

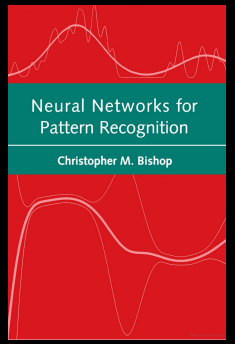
Implemente o algoritmo de treinamento para o perceptron no Google Colab.

Exercícios

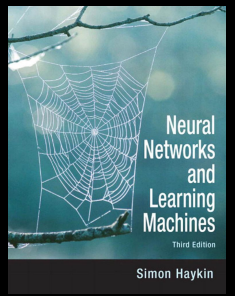
1. Considere que **w** é inicializado aleatoriamente, e que o problema em questão é linearmente separável. Executamos o algoritmo do perceptron duas vezes para o mesmo conjunto de treinamento. É possível que:
    a. O número de épocas necessárias para convergir para a solução seja diferente entre as duas execuções?
    b. A fronteira (valor final de **w**) seja diferente entre as duas execuções?
2. Faça um treino com as seguintes learning rates: 0.1, 0.5, 0.01, 0.05. Quantas épocas foram necessárias para o perceptron convergir? Qual seria a melhor learning rate testada e por quê? Dica: Desabilite a visualização da fronteira de decisão para diminuir o tempo de execução.
3. Modifique o problema do exemplo Iris Setosa para manter todas as quatro características do problema original, mas mantenha só as duas primeiras classes, como feito em aula.
4. Modifique o problema Iris Setosa agora para considerar todas as quatro características e três classes. Como usar múltiplos perceptrons em um *pipeline* para gerar fronteiras nesse problema multiclasse?

Referências

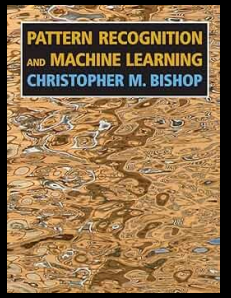
Bishop, C. M. Neural networks for pattern recognition. 1995.



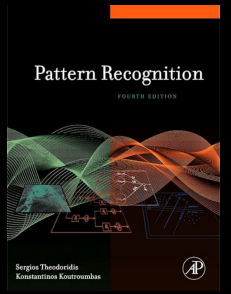
Haykin, S. S. Neural networks and learning machines. 2009.



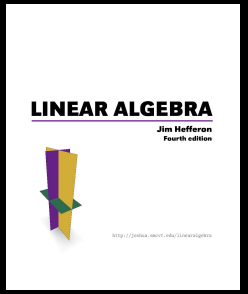
Bishop, C. M. Pattern Recognition and Machine Learning. 2006.



Theodoridis, S., Koutroumbas, K. Pattern Recognition & Matlab Intro. 2010.



Hefferon, J. Linear Algebra. 2015.



Licença