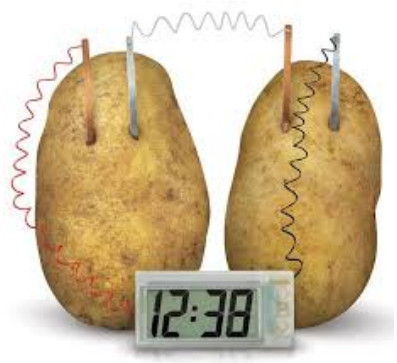
The perceptron

TEAM POTATO CLOCK

Erin Edkins
Alex Ludert
Gautier PICOT

ICS 635
University of Hawai'i
September 2, 2016



The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$

The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$

The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$
- Initialize the *weight* vector $w = (0, 0, 0)$

The Perceptron Algorithm

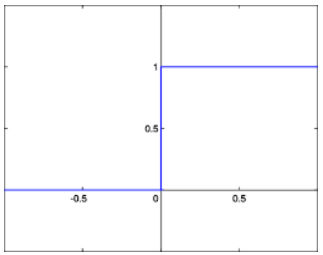
Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$
- Initialize the *weight* vector $w = (0, 0, 0)$
- Initialize the *outputs* $y_j = 1,\ 1 \leq j \leq N$

The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

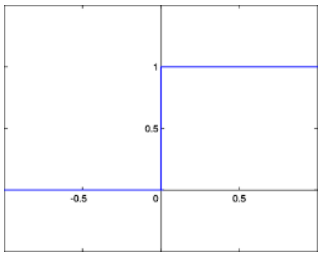
- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$
- Initialize the *weight* vector $w = (0, 0, 0)$
- Initialize the *outputs* $y_j = 1,\ 1 \leq j \leq N$
- Transfer θ : *Step function*



The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$
- Initialize the *weight* vector $w = (0, 0, 0)$
- Initialize the *outputs* $y_j = 1,\ 1 \leq j \leq N$
- Transfer θ : *Step function*

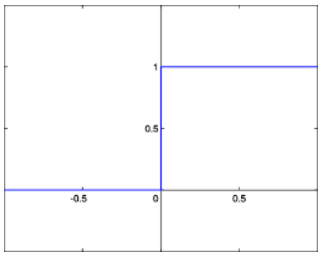


- While the classification is not achieved
  - Initialize `wrong`$= 0$
  - For each point P_j, calculate the *new label* $y_j = \theta(w \cdot P_j)$
  - If $y_j \neq l_j$, *update the weight* $w+ = c * y_j * P_J$ and `wrong`$+ = 1$

The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$
- Initialize the *weight* vector $w = (0, 0, 0)$
- Initialize the *outputs* $y_j = 1,\ 1 \leq j \leq N$
- Transfer θ : *Step function*

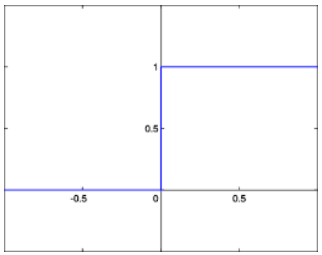


- While the classification is not achieved
  - Initialize `wrong`$= 0$
  - For each point P_j, calculate the *new label* $y_j = \theta(w \cdot P_j)$
  - If $y_j \neq l_j$, *update the weight* $w+ = c * y_j * P_J$ and `wrong`$+ = 1$
- Classification is *achieved* if `wrong`$= 0$

The Perceptron Algorithm

Our Perceptron program classifies of a set of N *two-dimensional* points (x_j, y_j) with label $l_j = \pm 1$ by using a *learning rate* $< c \leq 1$ (0.5 by default).

- *Datas* : $(x_j, y_j, l_j),\ 1 \leq j \leq N$
- *Points* : $P_j = (1, x_j, y_j),\ 1 \leq j \leq N$
- Initialize the *weight* vector $w = (0, 0, 0)$
- Initialize the *outputs* $y_j = 1,\ 1 \leq j \leq N$
- Transfer θ : *Step function*



- While the classification is not achieved
  - Initialize `wrong`$= 0$
  - For each point P_j, calculate the *new label* $y_j = \theta(w \cdot P_j)$
  - If $y_j \neq l_j$, *update the weight* $w+ = c * y_j * P_J$ and `wrong`$+ = 1$
- Classification is *achieved* if `wrong`$= 0$

  Remark : to guarantee that the algorithm stops, we *limit the while loop loops* (at most 50 iterations by default).

The Data Faker Generator

To test our algorithm, we have implemented a *Data Fake Generator.*

The Data Faker Generator

To test our algorithm, we have implemented a *Data Fake Generator.*

- Generates a cloud of random 2D points separated by a line

The Data Faker Generator

To test our algorithm, we have implemented a *Data Fake Generator.*

- Generates a cloud of random 2D points separated by a line
  - Random *slope* $-1 \leq m \leq 1$ and random y-intercept $-1 \leq b \leq 1$ by default.

The Data Faker Generator

To test our algorithm, we have implemented a *Data Fake Generator.*

- Generates a cloud of random 2D points separated by a line
  - Random *slope* $-1 \leq m \leq 1$ and random y-intercept $-1 \leq b \leq 1$ by default.
  - The user may enter desired values for m and b.

The Data Faker Generator

To test our algorithm, we have implemented a *Data Fake Generator.*

- Generates a cloud of random 2D points separated by a line
  - Random *slope* $-1 \leq m \leq 1$ and random y-intercept $-1 \leq b \leq 1$ by default.
  - The user may enter desired values for m and b.

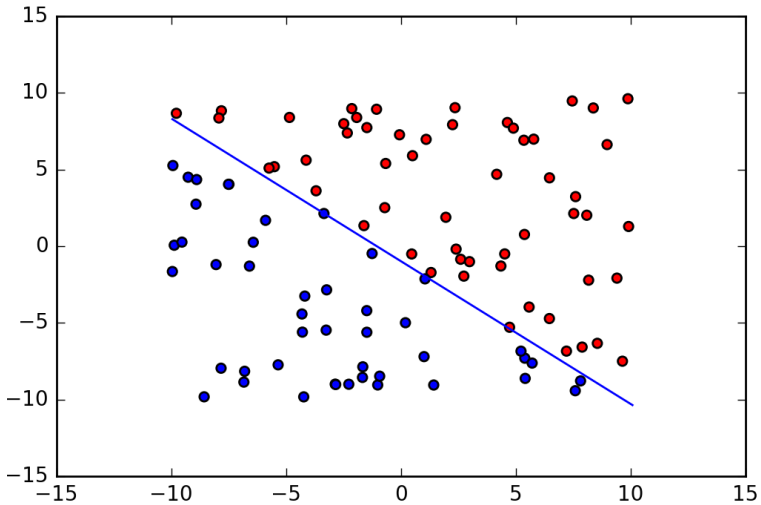
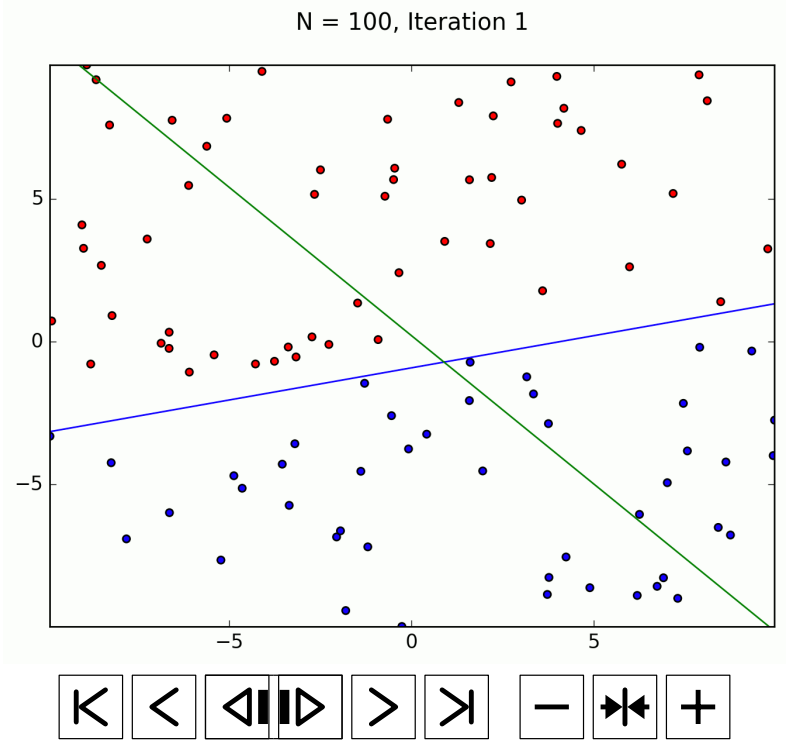


Figure : A collection 100 points linearly separated

Convergence of the algorithm : the linearly separable case

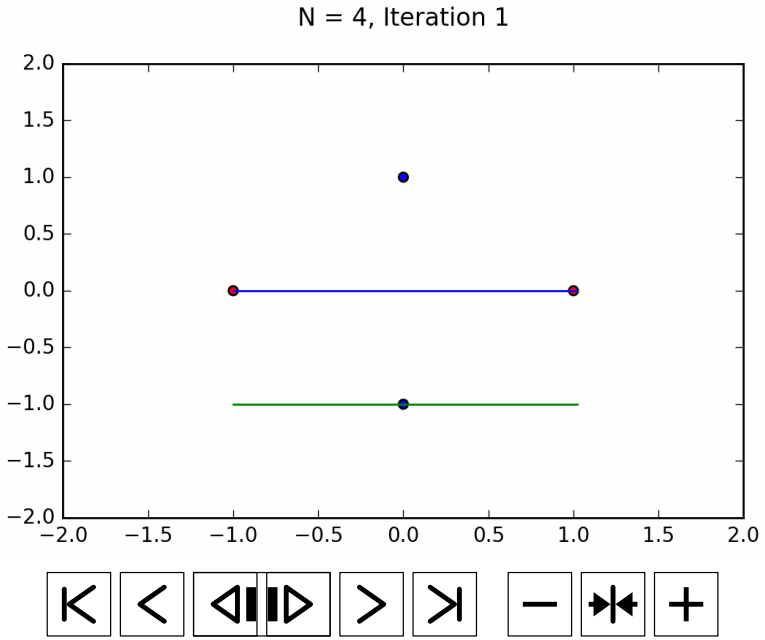
The algorithm *converges* when the classified datas can be separated by a line.



In this animation, the *green line* shows the line defined by the weight vector w. Through the execution of the algorithm, the line *stabilizes* $\iff$ the weight vector is *no longer* updated $\iff$ the algorithm *converges*.

Divergence of the algorithm : the non linearly separable case

The algorithm *diverges* when the classified datas can not be separated by a line.



In this animation, the *green line* shows the line defined by the weight vector w. Through the execution of the algorithm, the line *does not stabilize* $\iff$ the weight vector *keeps being updated* $\iff$ the algorithm *diverges*.

Thank you...

...Mister Rosenblatt !