PERCEPTRON

E. Fersini

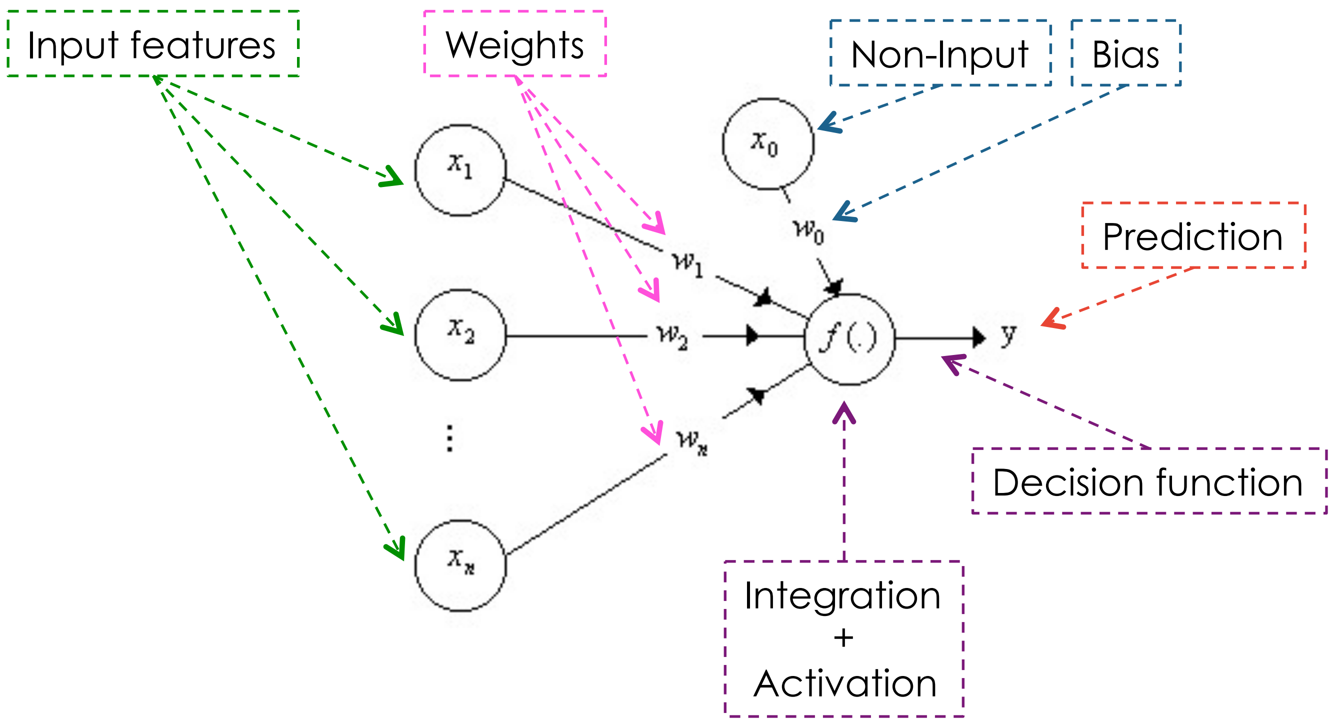
INTRODUCTION

- Unlike decision trees, used in in the previous lab, the process of how neural networks (and support vector machines) transforms from input to output is less clear and can be hard to interpret.
  - Both are usually used as black boxes.

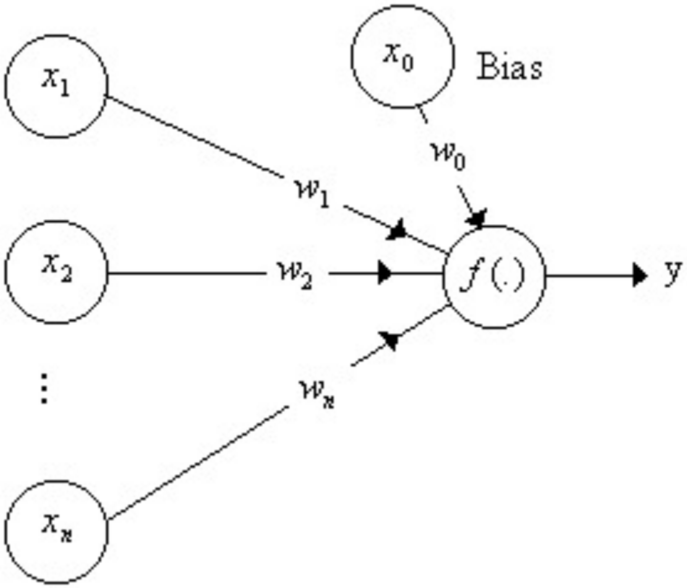
- The development of a neural network is inspired by human brain activities. As such, this type of network is a computational model that mimics the pattern of the human mind.

PERCEPTRON

- A perceptron takes a vector of inputs $x = (x_1, x_2, \ldots, x_n)$, weights each feature, and outputs a binary variable, "+1" or "-1", according to an activation function (i.e. depending on whether a weighted sum exceeds some pre-determined threshold).



PERCEPTRON



Activation

Integration

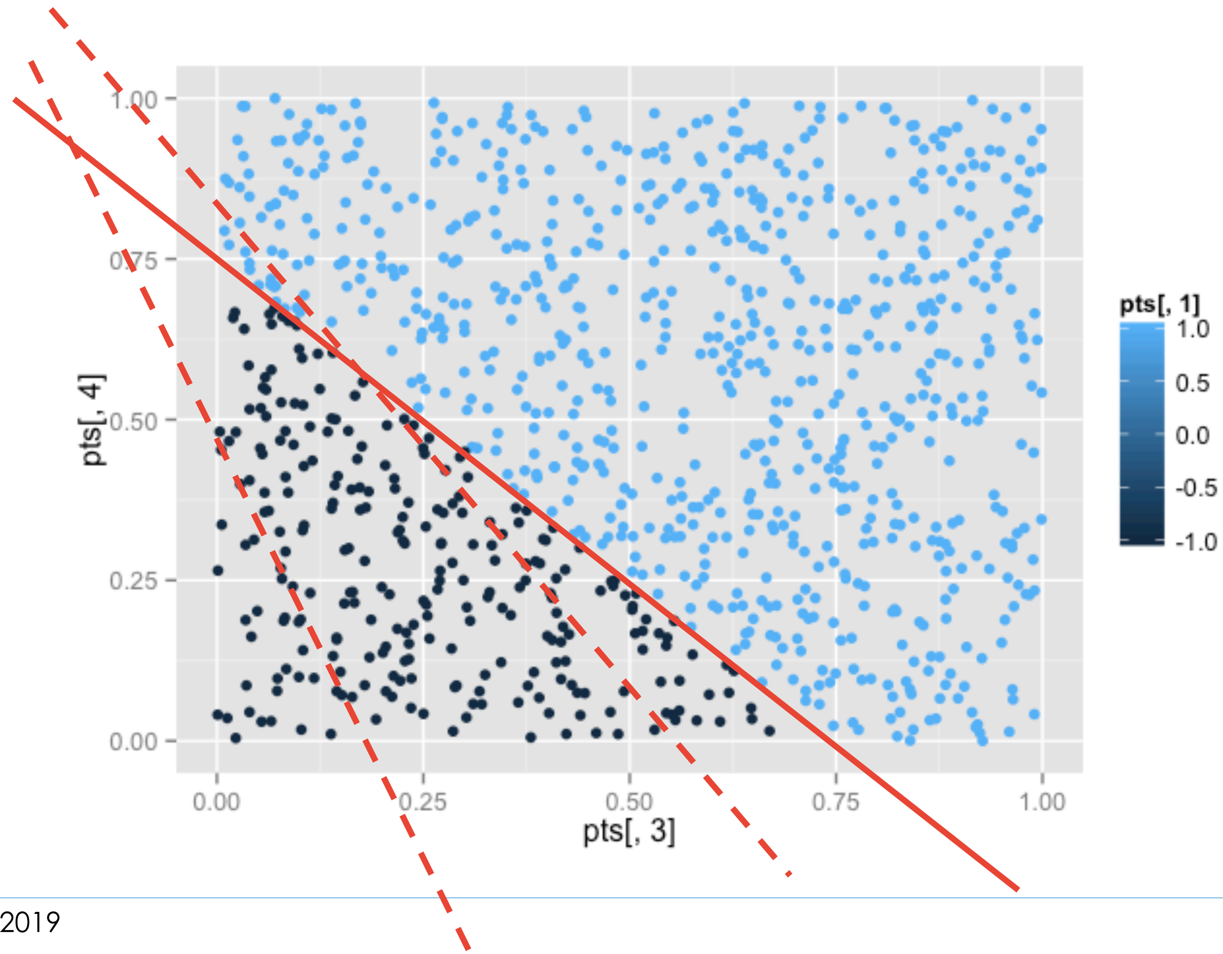
$$f(x,w) = \underbrace{sign}_{\text{Activation}}(\underbrace{w_1 * x_1 + \ldots + w_n * x_n + w_0 * x_0}_{\text{Integration}})$$

$x_0=1$; w_0 alters the position of the decision boundary

Remark: If w_0 is negative, then the weighted combination of inputs must produce a positive value greater than $|b|$ to push the classifier over the 0 threshold

PERCEPTRON

- This means that w_0 is (usually) a paramter to be estimated



GENERATE THE DATASET

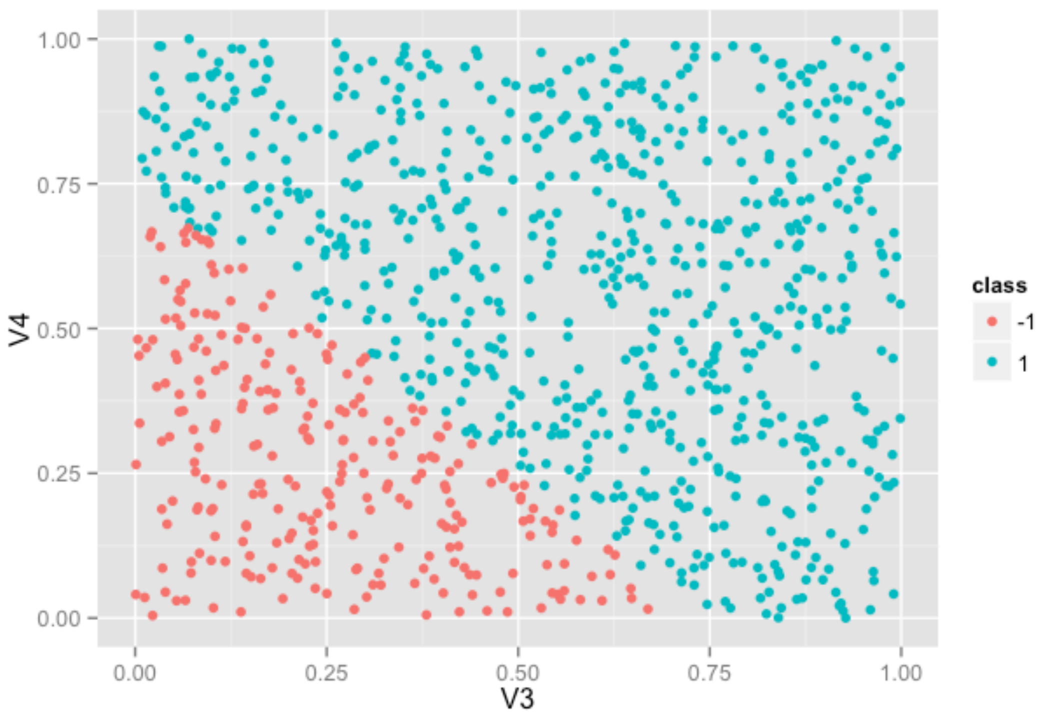
- We can generate a two dimensions dataset just to understand the perceptron

```
> Random.Unit = function(n, dim, threshold) {
    points <- runif(n * dim)
    points <- matrix(points, ncol = dim)
    label <- ifelse(apply(points, 1, sum) < threshold, -1, 1)
    return(cbind(label, x0 = rep(1, n), points))
}
```

```
> dataset = Random.Unit(1000, 2, 0.75)
```

DATASET EXPLORATION

```
> library(ggplot2)
> qplot(dataset[,3], dataset[,4],  colour = factor(dataset[,1]),
xlab="V3", ylab="V4") + labs(colour = 'class')
```



PERCEPTRON

- Today we will start by implementing the Perceptron algorithm:

```
function(data, threshold) {
   1. Initialize the weights: w0=-threshold, w1,w2 random
   2. Set a control variable for convergence
   3. Check convergency: all instances shluld be correctly classified
       3a. Adjust weights for misclassified instances
   4. return weights
}
```

To verify that an instance is misclasified

Label	Prediction	Misclassification
+1	+1	OK
+1	-1	error
-1	+1	error
-1	-1	OK

If an instance is misclassified adjust weights

$$w_i = w_i + label_i * input_i$$

PERCEPTRON

- Today we will start by implementing the Perceptron algorithm:

```
function(data, threshold) {
    1. Initialize the weights: w0=-threshold, w1,w2 random
    2. Set a control variable for convergence
    3. Check convergency: all instances shluld be correctly classified
        3a. Adjust weights for misclassified instances
    4. return weights
}
```

To verify that an instance is misclasified

Label	Prediction	Misclassification
+1	+1	OK
+1	-1	error
-1	+1	error
-1	-1	OK

If an instance is misclassified adjust weights

- If *x(t)* is correctly classified nothing changes.
- If *x(t)* is misclassified as negative, then *y(t)=1*. It follows that the new dot product increased by *x(t)* ·*x(t)* that is positive.
- If *x(t)* is misclassified as positive, then *y(t)=-1*. It follows that the new dot product decreased by *x(t)* ·*x(t)* that is positive.

PERCEPTRON

- Today we will start by implementing the Perceptron algorithm:

```
> function(data, threshold) {
    .........
    .........
    .........
    .........
    .........
    .........

    return(w)
}
```

PERCEPTRON

- Today we will start by implementing the Perceptron algorithm:

```
Classify <- function(x, weights) {
..................#only one instruction!
}
```

PERCEPTRON

- Today we will start by implementing the Perceptron algorithm:

```
> function(data, eta) {
   .........
   .........
   .........
   .........
   .........
   .........

   return(w)
}
```

Now you should define the **DELTA RULE**, assuming a **sigmoid** as output function

Show the behaviour of the perceptron using different **eta**