COMP24111 lecture 3

The Linear Classifier,

also known as the “Perceptron”

LAST WEEK: our first "machine learning" algorithm

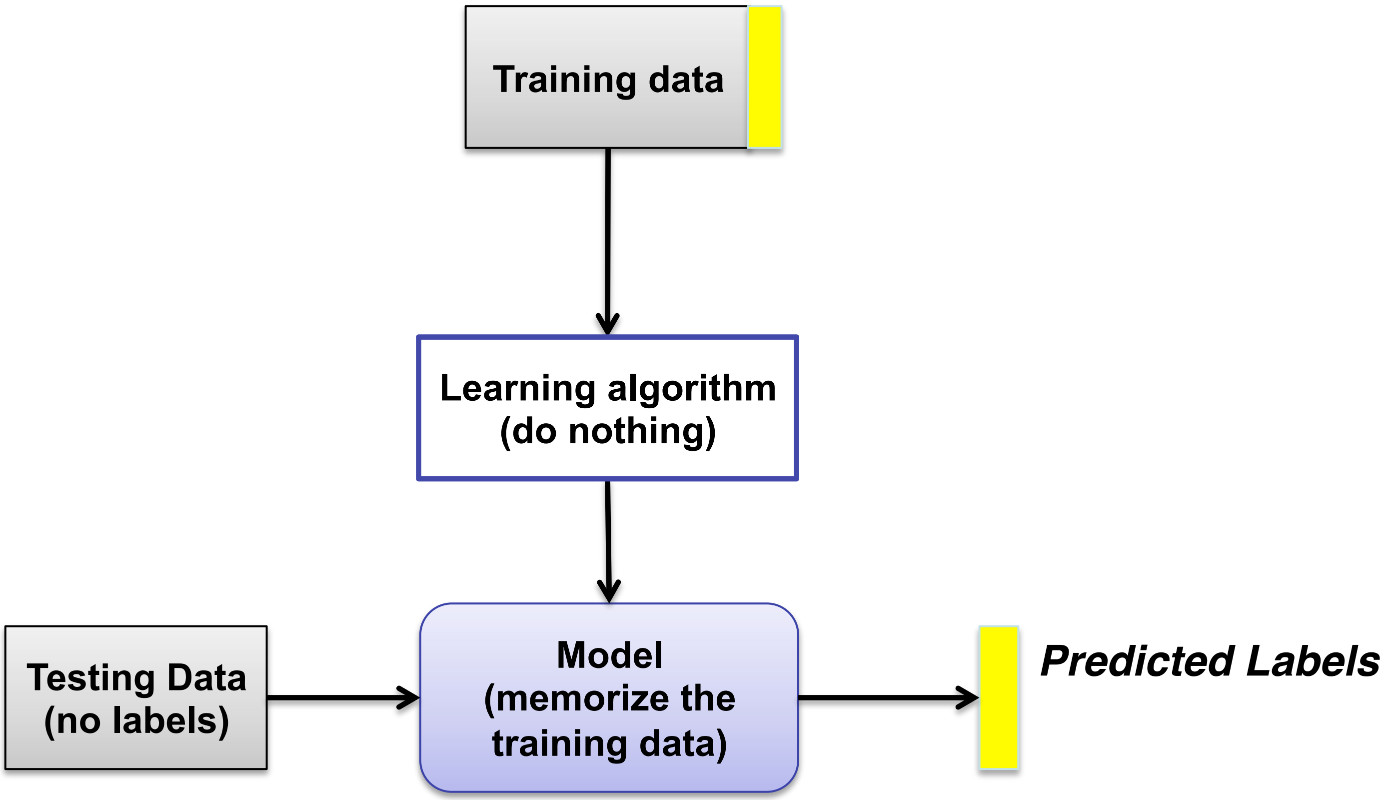
The K-Nearest Neighbour Classifier

```
Testing point x
For each training datapoint x'
   measure distance(x,x')
End
Sort distances
Select K nearest
Assign most common class
```

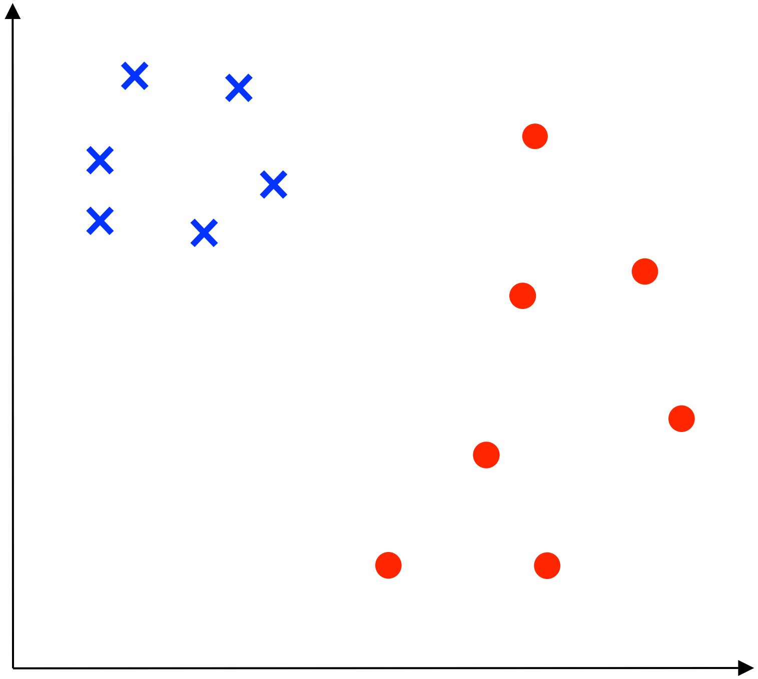
Make your own notes on its advantages / disadvantages.

I'll ask for volunteers next time we meet…..

Supervised Learning Pipeline for Nearest Neighbour

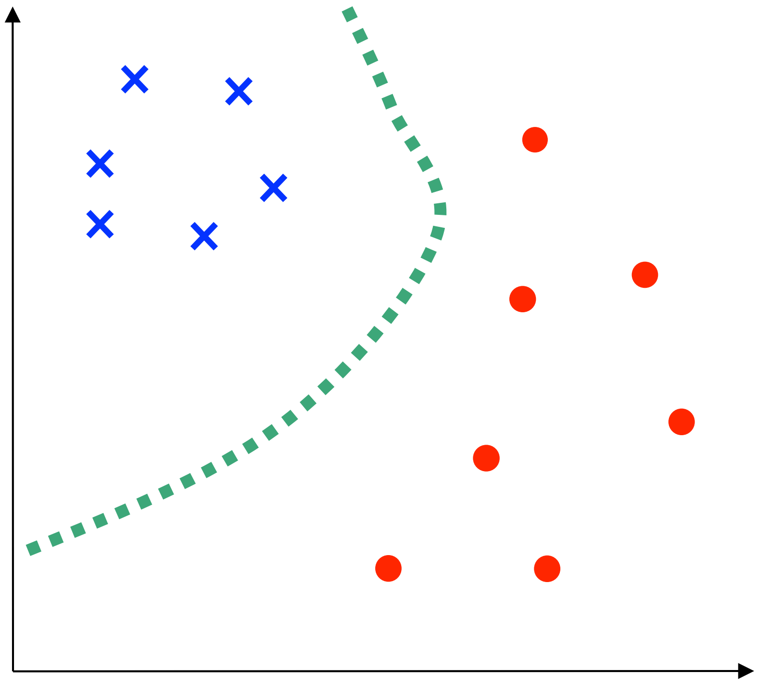


The *most important* concept in Machine Learning



The *most important* concept in Machine Learning

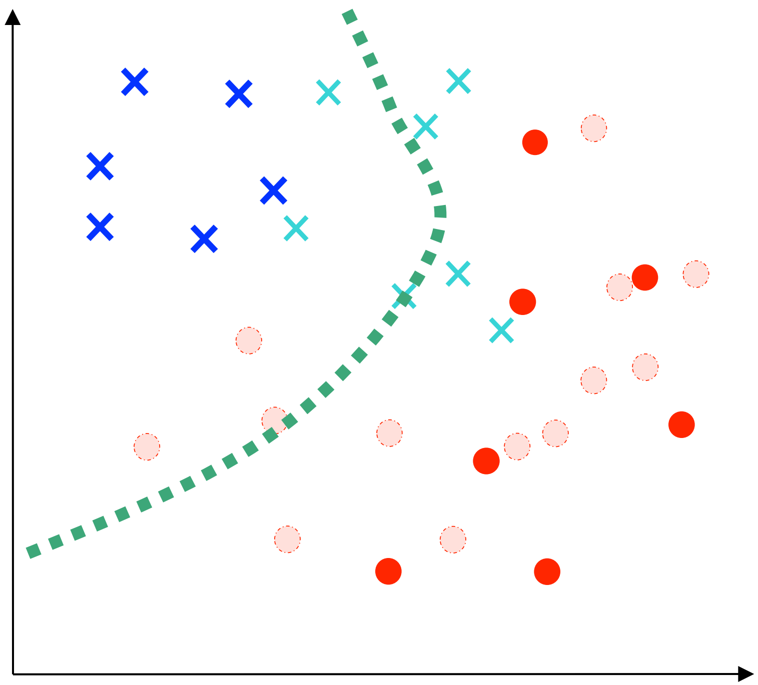
Looks good so far…



The *most important* concept in Machine Learning

Looks good so far…

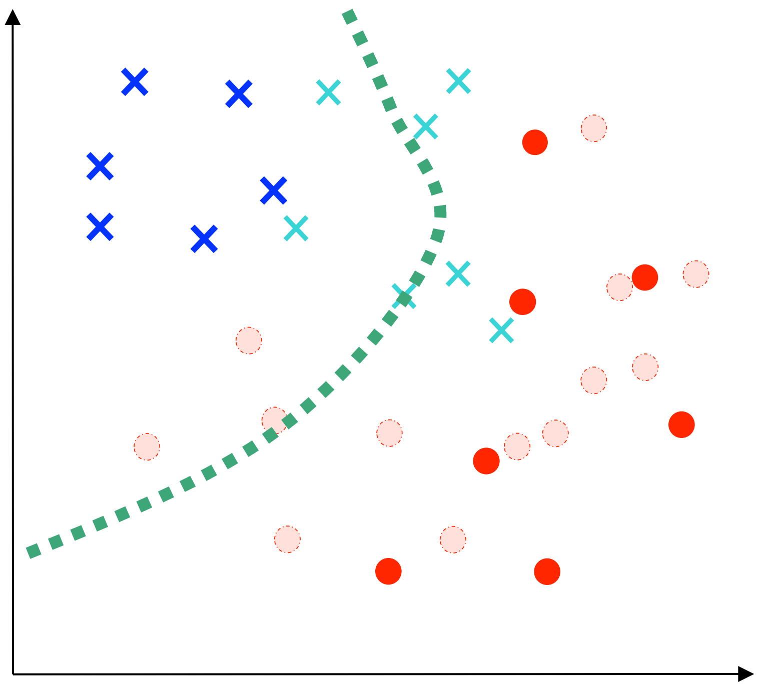
Oh no! Mistakes!
What happened?



The *most important* concept in Machine Learning

Looks good so far…

Oh no! Mistakes!
What happened?



We didn't have all the data.

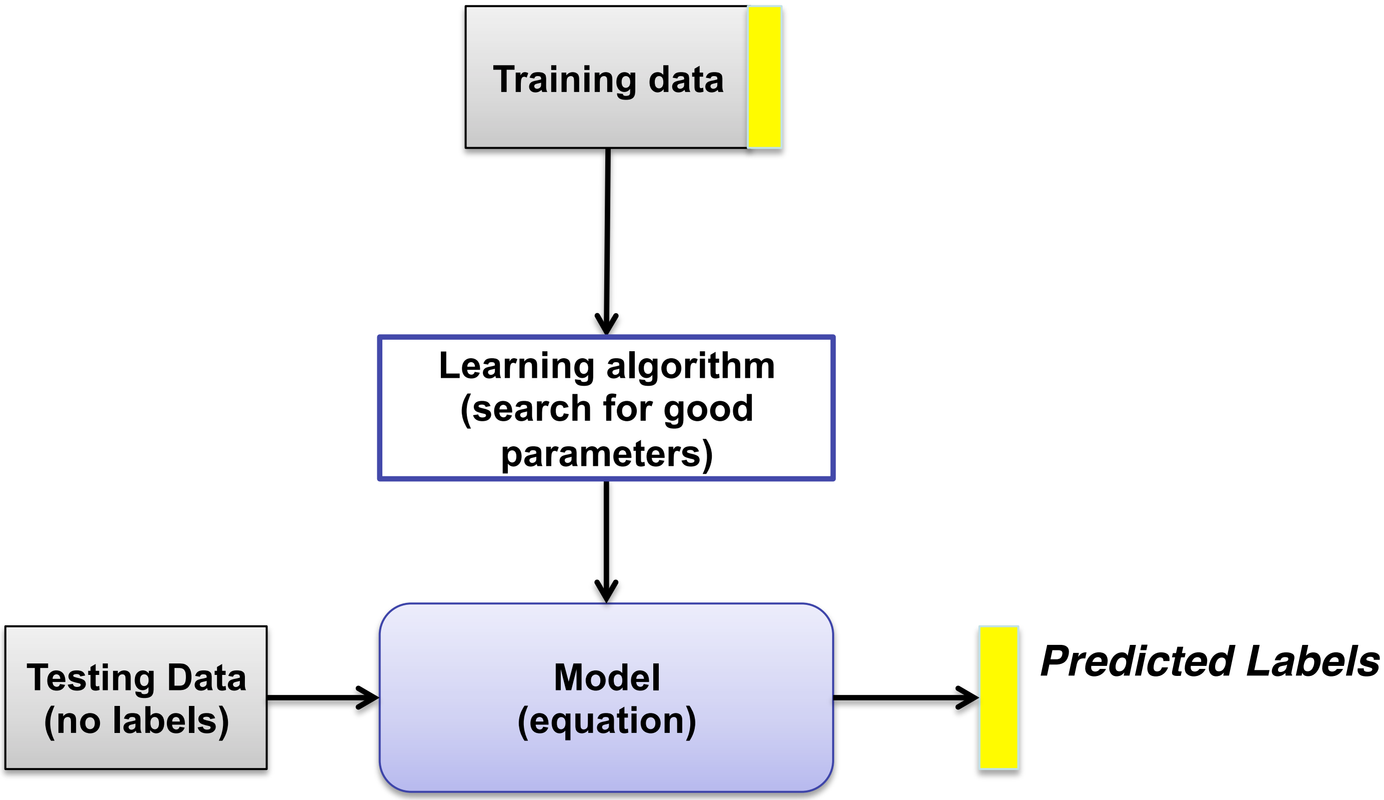
We can never assume that we do.

This is called "OVER-FITTING"
to the small dataset.

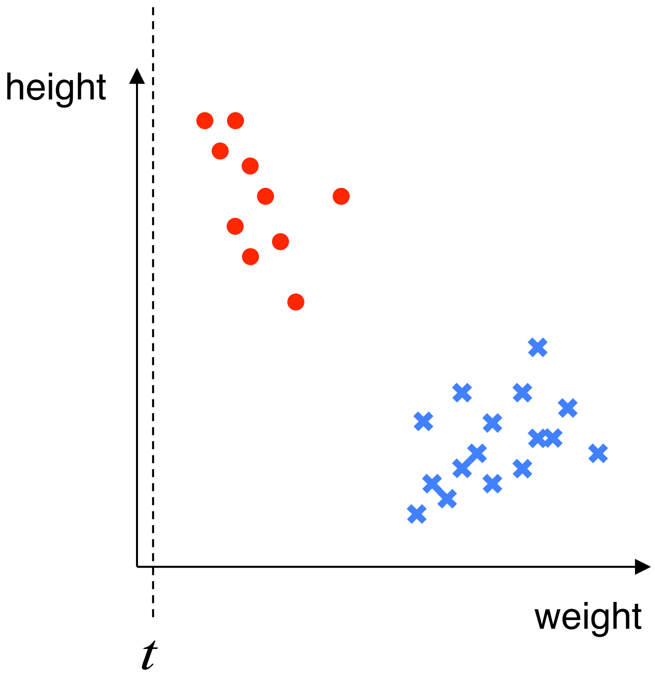
COMP24111 lecture 3

The Linear Classifier

Supervised Learning Pipeline for Linear Classifiers



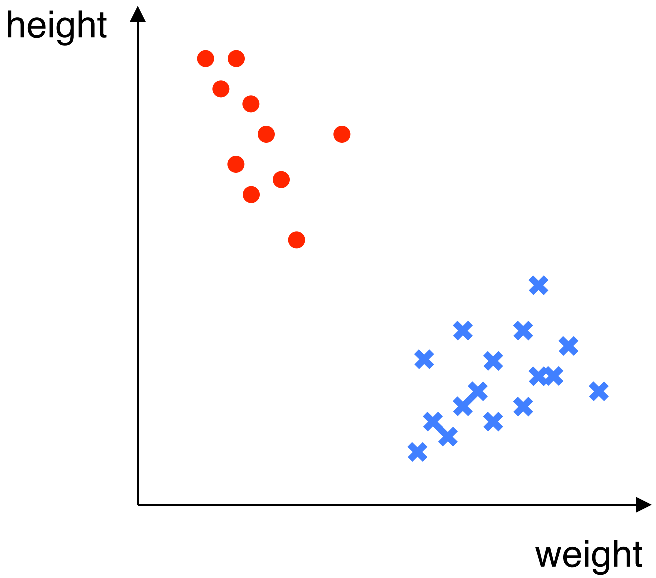
A more simple, *compact* model?



$$\text{if } (\mathit{weight} > t) \text{ then "player" else "dancer"}$$

What's an algorithm to find a good threshold?

```
t=40
while ( numMistakes != 0 )
{
        t = t + 1
        numMistakes = testRule(t)
}
```



$$\text{if } (weight > t) \text{ then "player" else "dancer"}$$

We have our second Machine Learning procedure.

The threshold classifier (also known as a "Decision Stump")

$$\text{if } (weight > t) \text{ then "player" else "dancer"}$$

```
t=40
while ( numMistakes != 0 )
{
        t = t + 1
        numMistakes = testRule(t)
}
```

Three “ingredients” of a Machine Learning procedure

“Model”	*The final product, the thing you have to package up and send to a customer. A piece of code with some parameters that need to be set.*
“Error function”	*The performance criterion: the function you use to judge how well the parameters of the model are set.*
“Learning algorithm”	*The algorithm that optimises the model parameters, using the error function to judge how well it is doing.*

Three "ingredients" of a Threshold Classifier

Error function

```
t=40
while ( numMistakes != 0 )
{
        t = t + 1
        numMistakes = testRule(t)
}
```

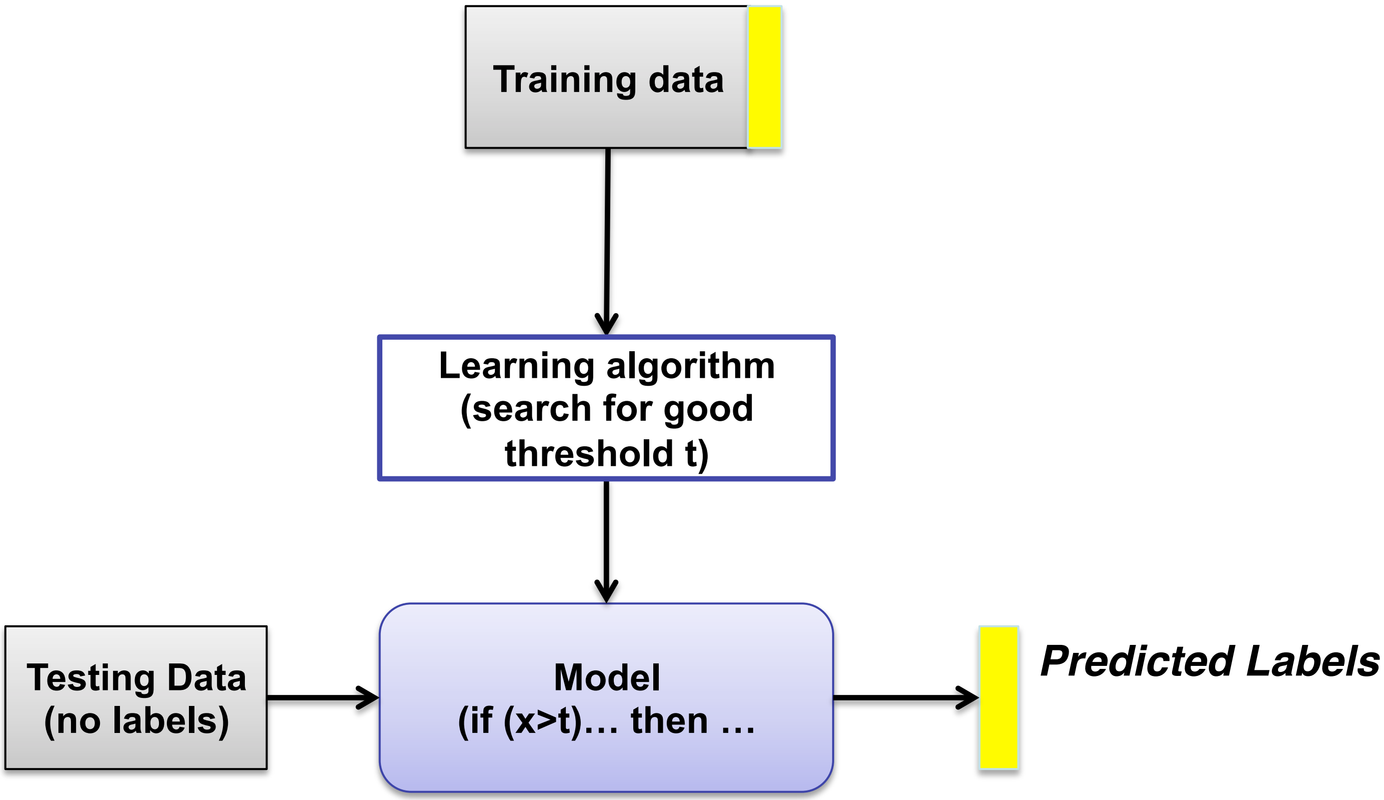
Learning algorithm

$$\text{if } (x > t) \text{ then "player" else "dancer"}$$

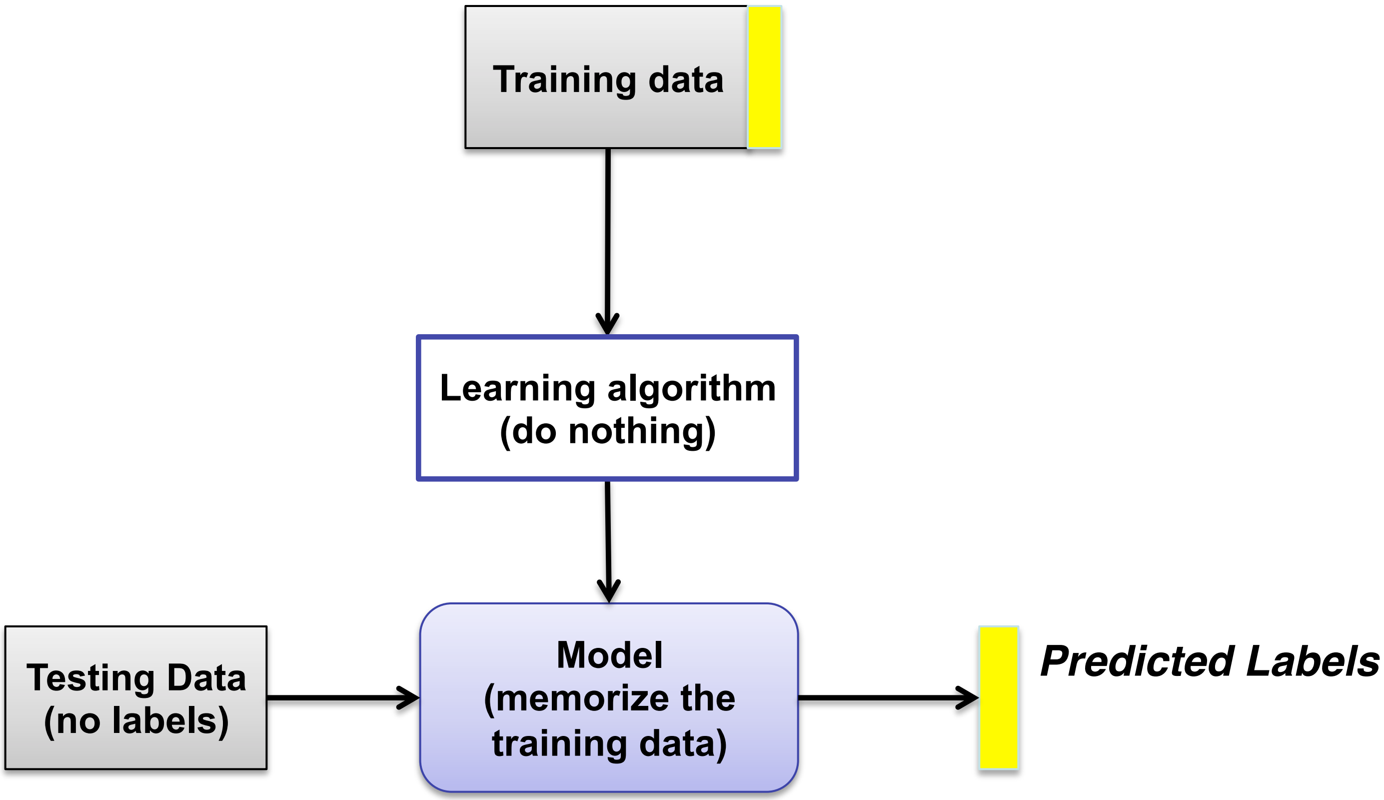
Model

General case – we're not just talking about the weight of a rugby player – a threshold can be put on any feature 'x'.

Supervised Learning Pipeline for Threshold Classifier



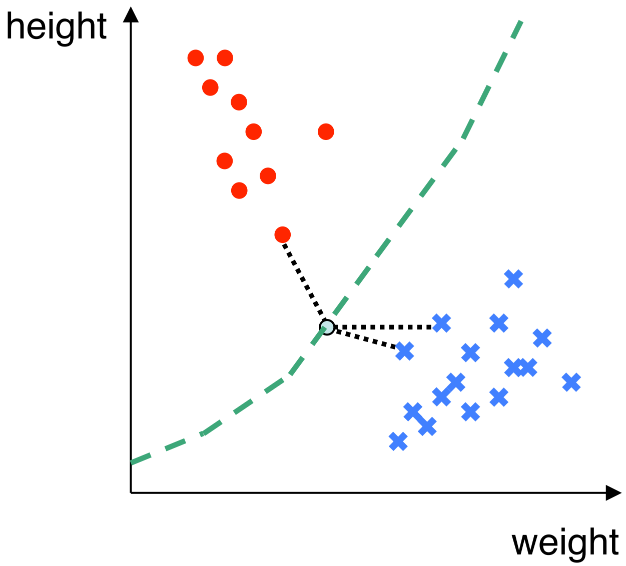
Supervised Learning Pipeline for Nearest Neighbour



What's the "model" for the Nearest Neighbour classifier?

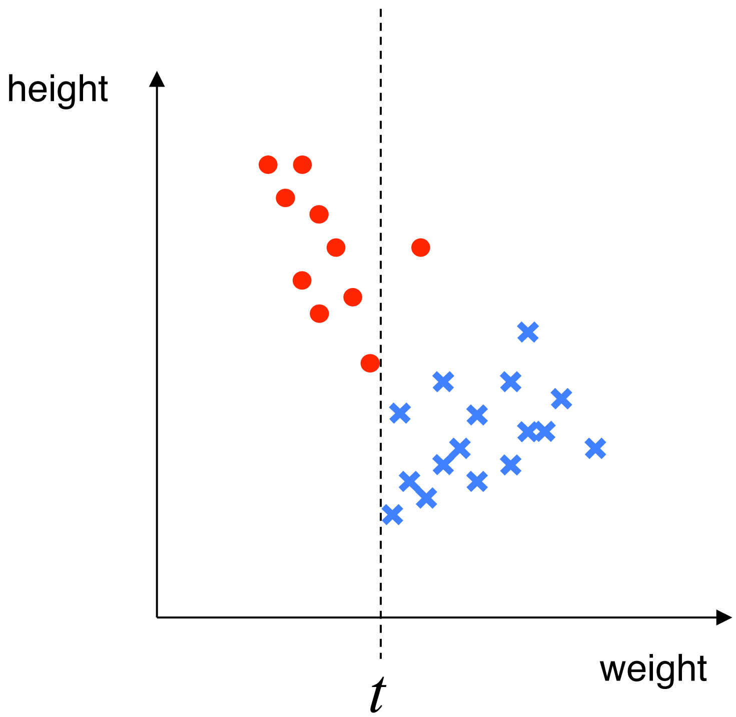
For the k-nn, the model is the *training data* itself !

- *very good accuracy* ☺
- *very computationally intensive!* ☹




```
Testing point x
For each training datapoint x'
   measure distance(x,x')
End
Sort distances
Select K nearest
Assign most common class
```

New data: what's an algorithm to find a good threshold?

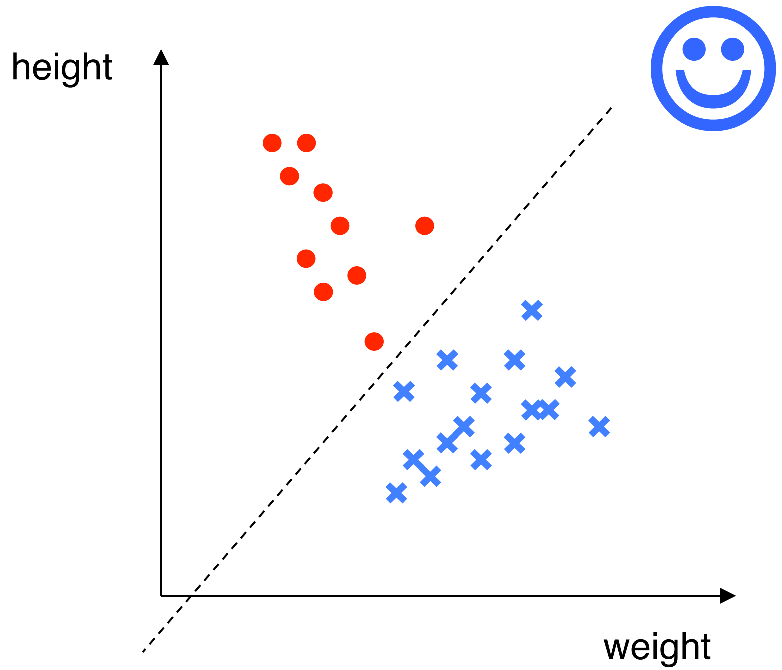


Our model does not match the problem!

$$\text{if } (weight > t) \text{ then "player" else "dancer"}$$

1 mistake…

New data: what's an algorithm to find a good threshold?

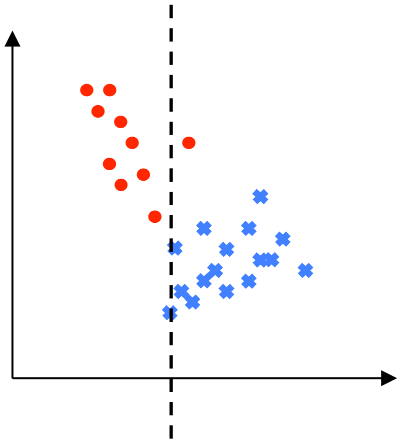


But our current model cannot represent this…

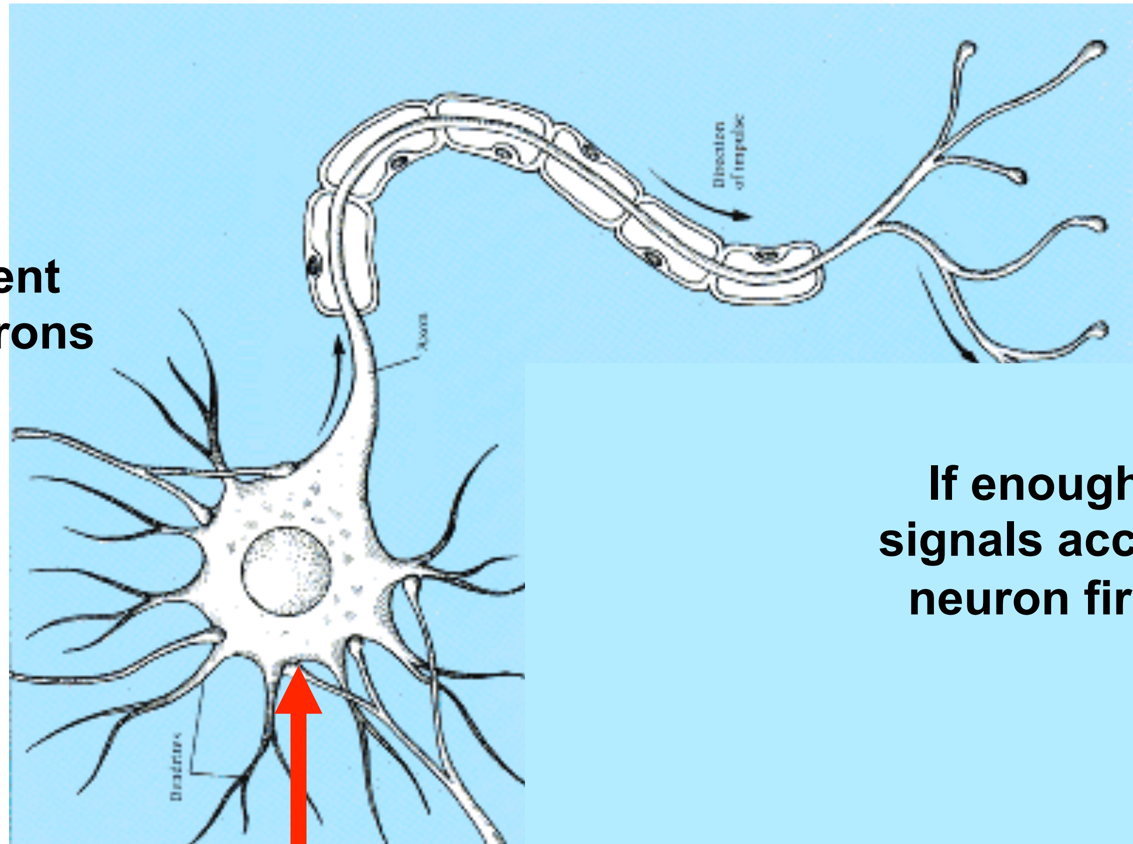
$$\text{if } (weight > t) \text{ then "player" else "dancer"}$$

We need a more sophisticated model…

$$\text{if } (x > t) \text{ then "player" else "dancer"}$$



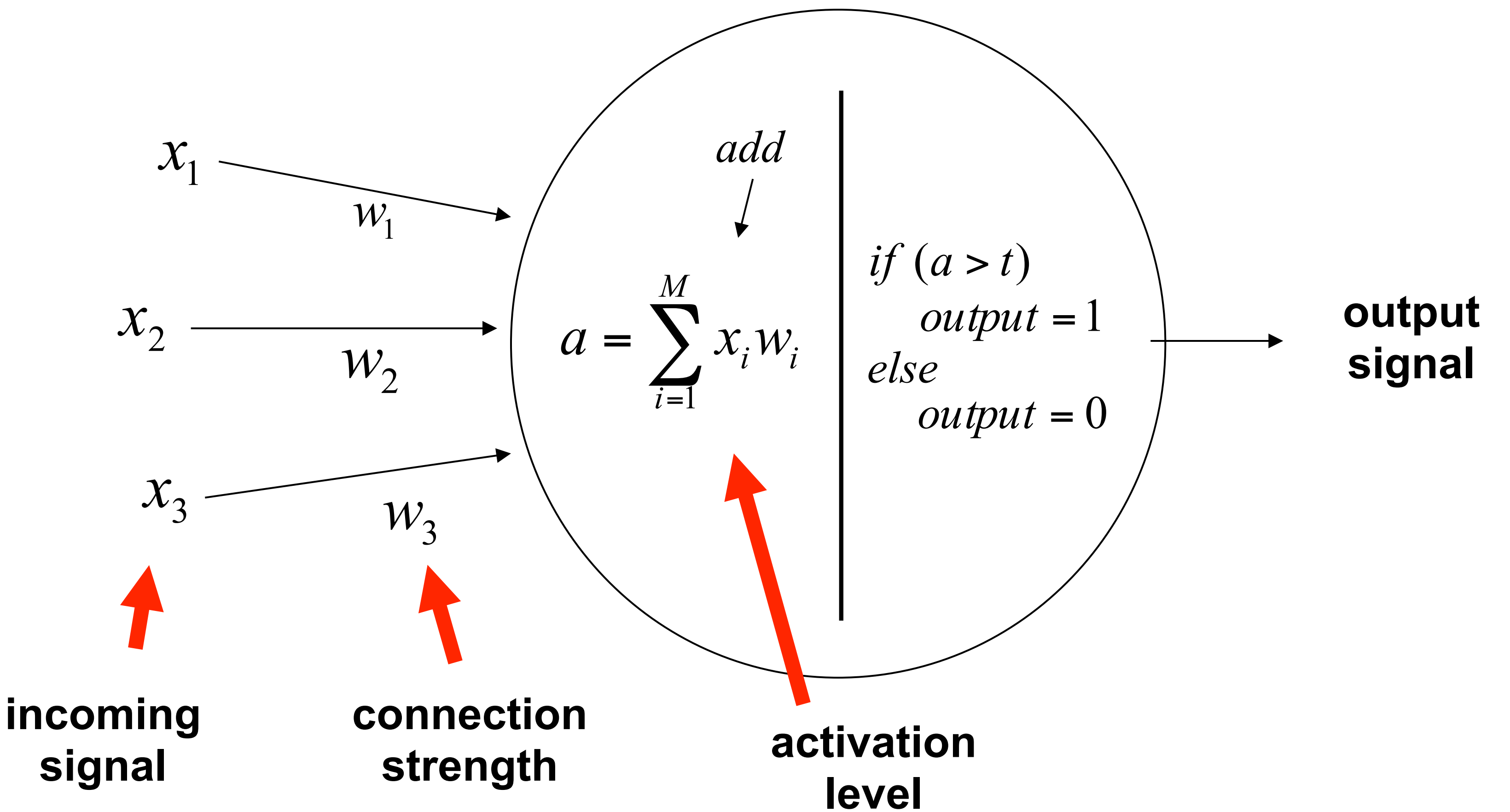
Input signals sent from other neurons



If enough sufficient signals accumulate, the neuron fires a signal.

Connection strengths determine how the signals are accumulated

- input signals ‘x’ and coefficients ‘w’ are multiplied
- weights correspond to connection strengths
- signals are added up – if they are enough, FIRE!



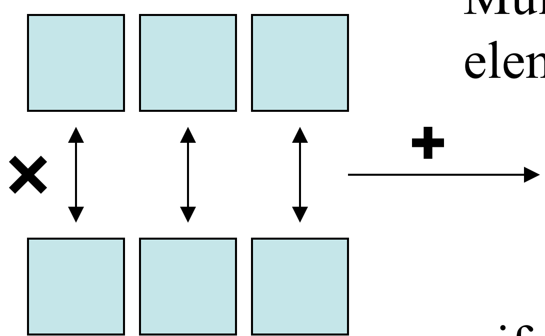
Calculation…

$$a = \sum_{i=1}^{M} x_i w_i$$

Sum notation

(just like a loop from 1 to M)

`double[] x =`



Multiple corresponding elements and add them up

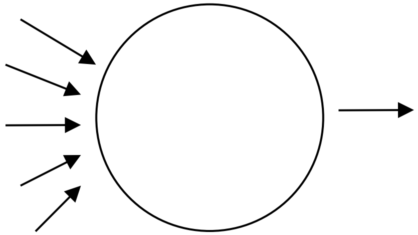
`a` (activation)

`double[] w =`

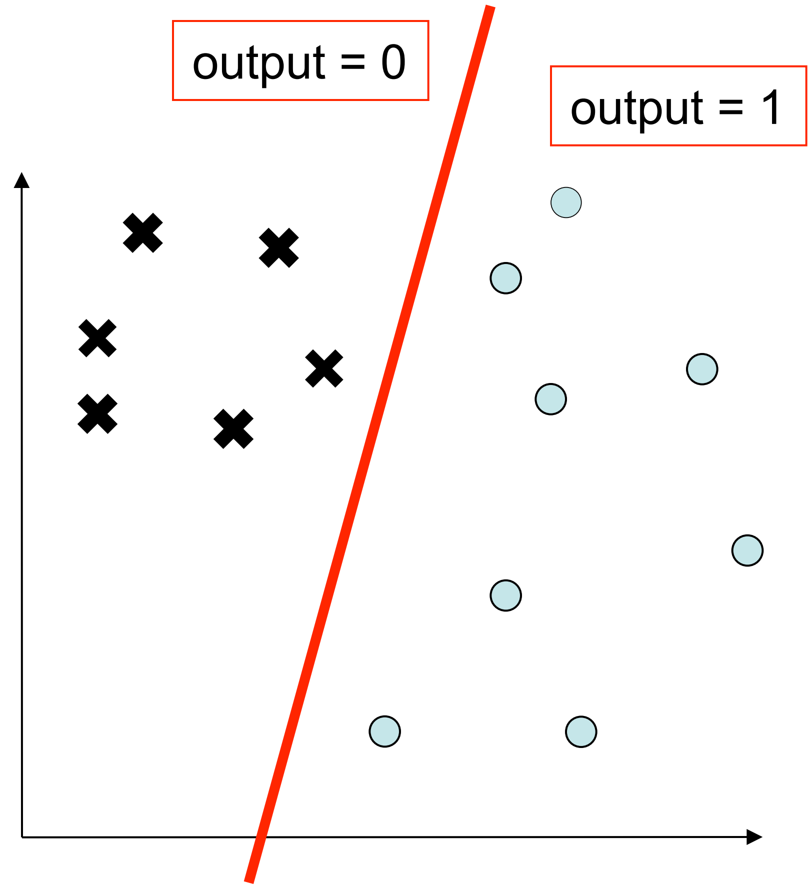
if (activation > threshold) FIRE !

The Perceptron Decision Rule

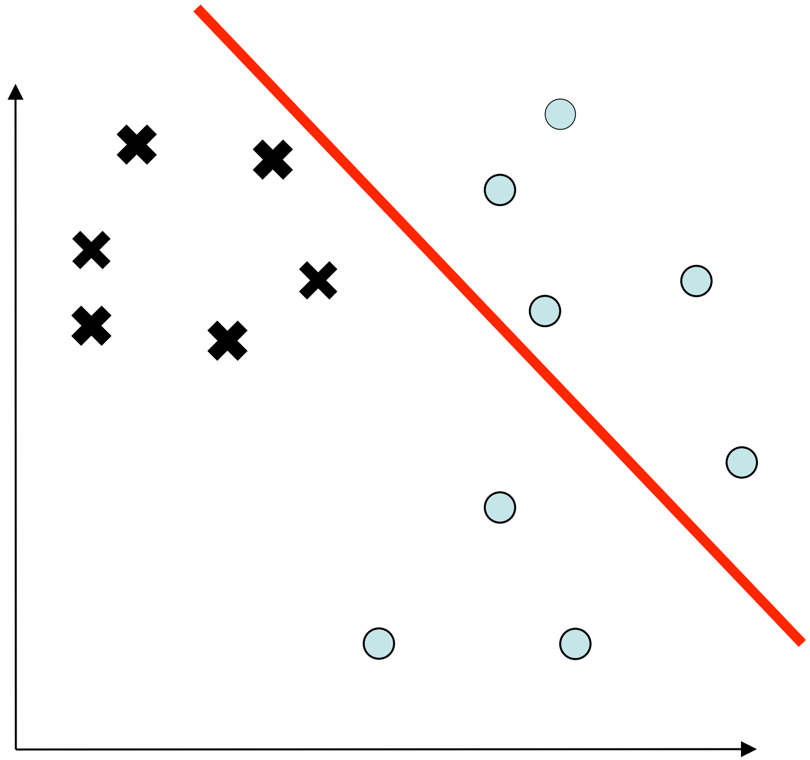
$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \qquad \text{then } output = 1, \text{ else } output = 0$$



$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

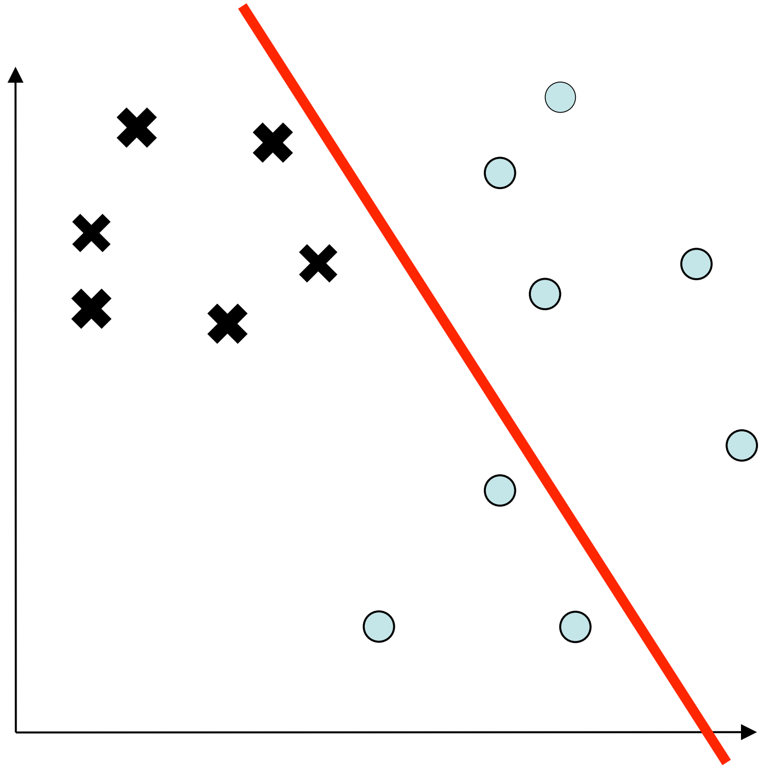


Rugby player = 1
Ballet dancer = 0



Is this a good decision boundary?

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

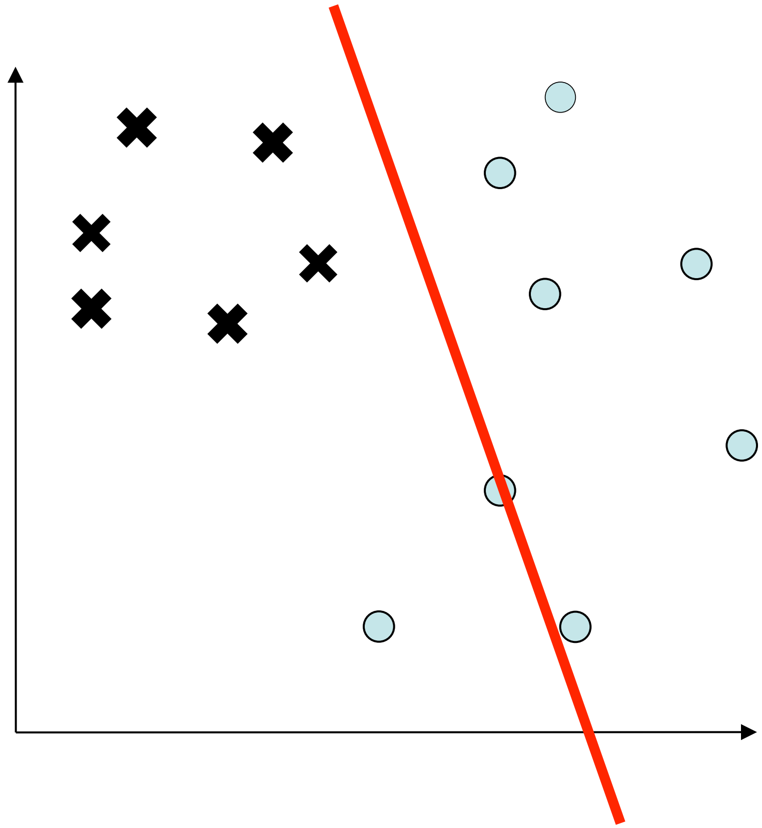


$w_1 = 1.0$

$w_2 = 0.2$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$

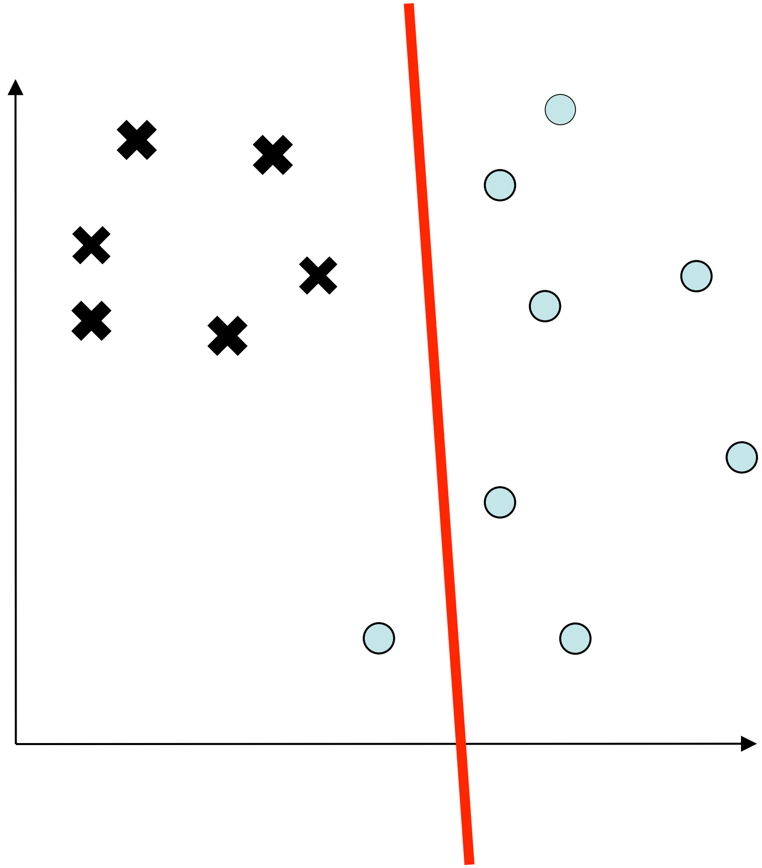


$w_1 = 2.1$

$w_2 = 0.2$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$

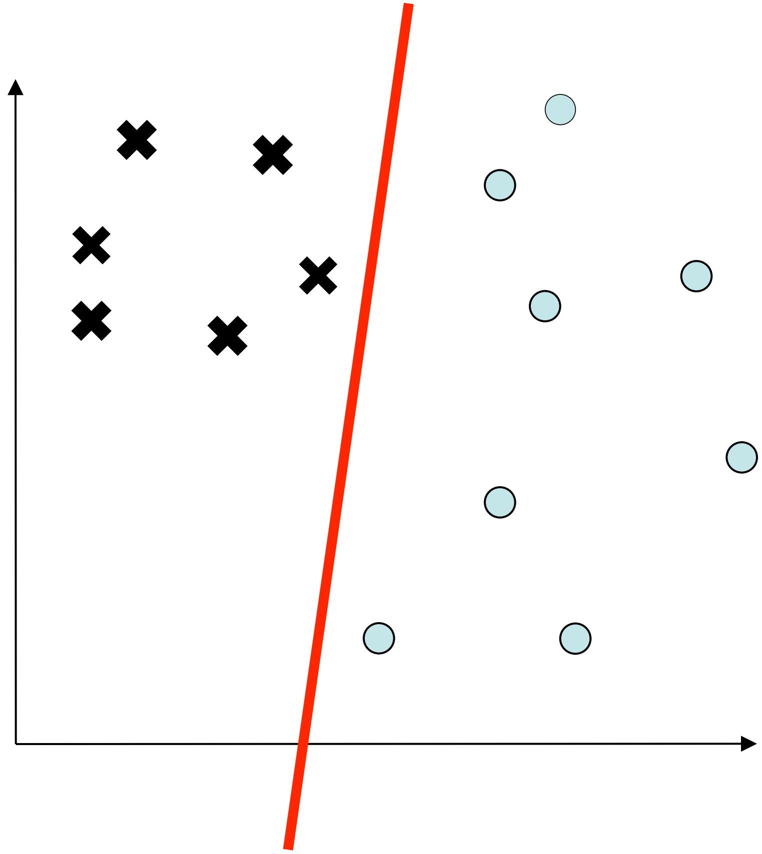


$w_1 = 1.9$

$w_2 = 0.02$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$



$w_1 = -0.8$

$w_2 = 0.03$

$t = 0.05$

Changing the weights/threshold makes the decision boundary move.

Pointless / impossible to do it by hand – only ok for simple 2-D case.

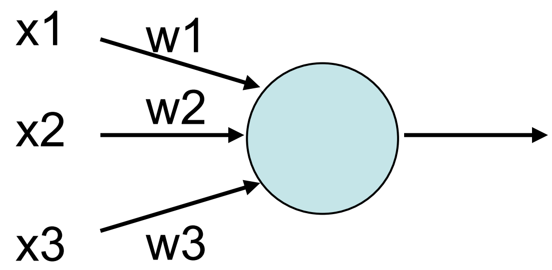
We need an algorithm….

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q1. What is the activation, ***a***, of the neuron?

1.0 * 0.2 + 0.5 * 0.5 + 2.0 * 0.5 = 1.45

Q2. Does the neuron fire?

1.45 > 1 --> FIRE!!!!

Q3. What if we set threshold at 0.5 and weight #3 to zero?

a = 0.45 --> 0.45 < 0.5 --> NO FIRE!!!!

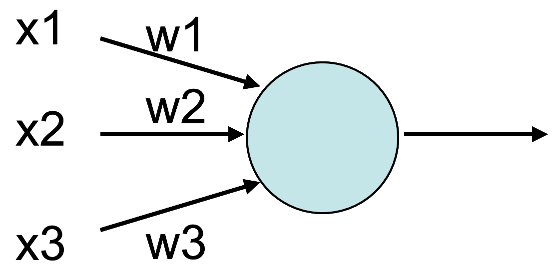
20 minute break

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q1. What is the activation, *a*, of the neuron?

$$a = \sum_{i=1}^{M} x_i w_i = (1.0 \times 0.2) + (0.5 \times 0.5) + (2.0 \times 0.5) = 1.45$$

Q2. Does the neuron fire?

if (activation > threshold) output=1 else output=0

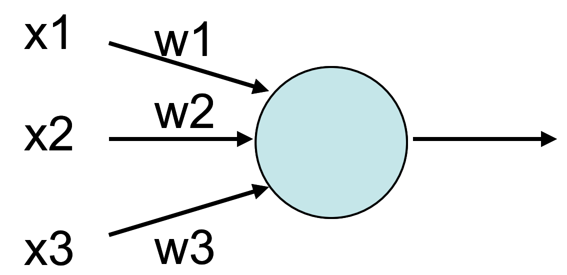
…. So yes, it fires.

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q3. What if we set threshold at 0.5 and weight #3 to zero?

$$a = \sum_{i=1}^{M} x_i w_i = (1.0 \times 0.2) + (0.5 \times 0.5) + (2.0 \times 0.0) = 0.45$$

if (activation > threshold) output=1 else output=0

…. So no, it does not fire..

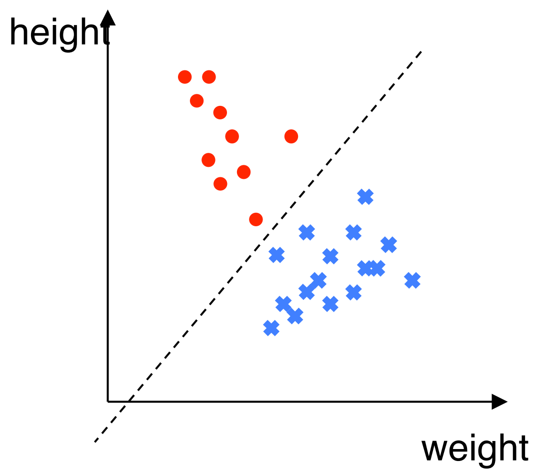
We need a more sophisticated model…

$$\text{if } (weight > t) \text{ then "player" else "dancer"}$$

$$\text{if } (f(\vec{x}) > t) \text{ then "player" else "dancer"}$$

$$x_1 = height\ (cm)$$

$$x_2 = weight\ (kg)$$



The Perceptron

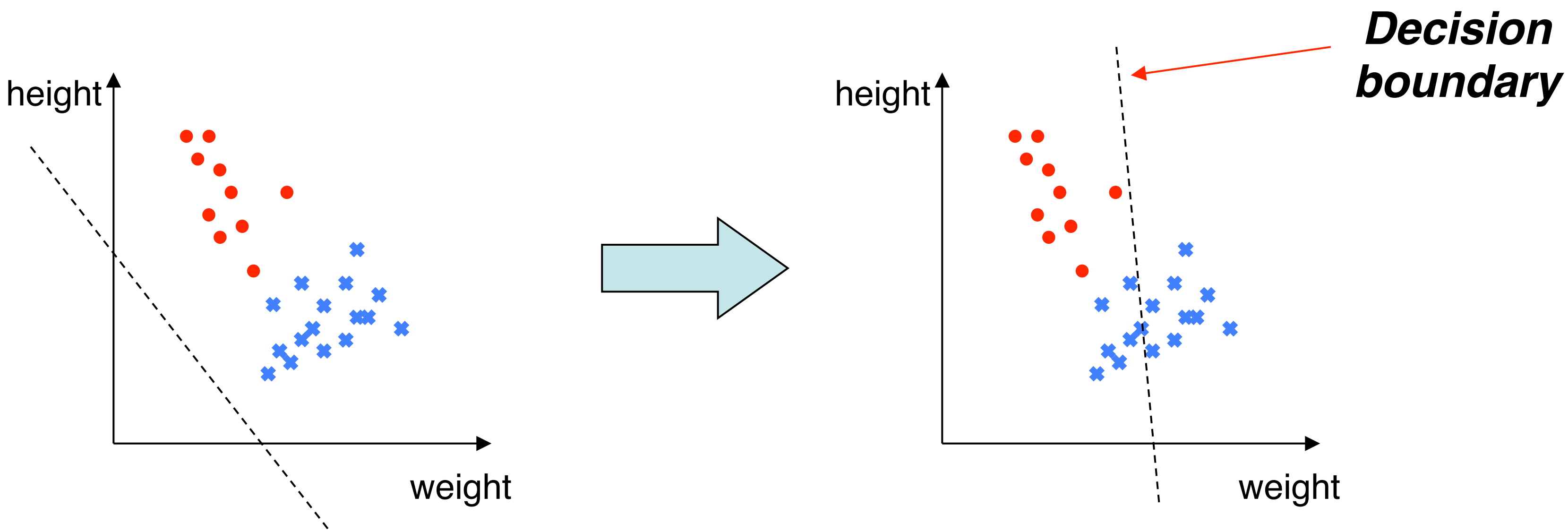
$$f(\vec{x}) = (w_1 * x_1) + (w_2 * x_2)$$

$$= \sum_{i=1}^{d} w_i x_i$$

The Perceptron

$$\text{if } f(\vec{x}) > t \text{ then "player" else "dancer"}$$

$$f(\vec{x}) = (w_1 * x_1) + (w_2 * x_2)$$
$$= \sum_{i=1}^{d} w_i x_i$$



w_1, w_2 **and** t **change the position of the <u>DECISION BOUNDARY</u>**

The Perceptron

Model

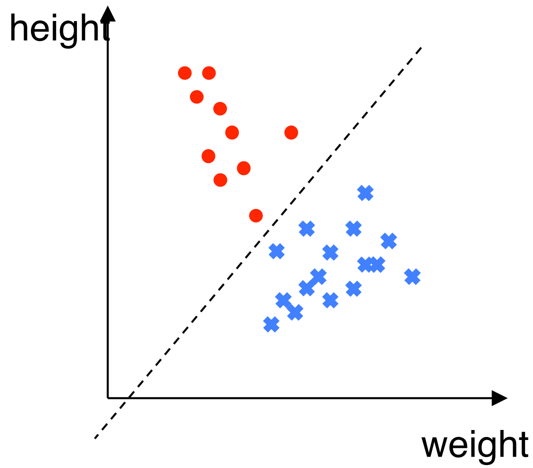
$$\text{if } \sum_{i=1}^{d} w_i x_i > t \text{ then } \hat{y} = 1 \text{ else } \hat{y} = 0 \quad \begin{cases} \textit{"player"} = 1 \\ \textit{"dancer"} = 0 \end{cases}$$

Error function

Number of mistakes (a.k.a. classification error)

Learning algo.

??? .... need to optimise the w and t values...



Perceptron Learning Rule

$$\text{new weight} = \text{old weight} + \underbrace{0.1 \times (\,\text{trueLabel} - \text{output}\,) \times \text{input}}_{\textit{update}}$$

What weight updates do these cases produce?

if… (**`target = 0`**, **`output = 0`**) …. then update = ?

if… (**`target = 0`**, **`output = 1`**) …. then update = ?

if… (**`target = 1`**, **`output = 0`**) …. then update = ?

if… (**`target = 1`**, **`output = 1`**) …. then update = ?

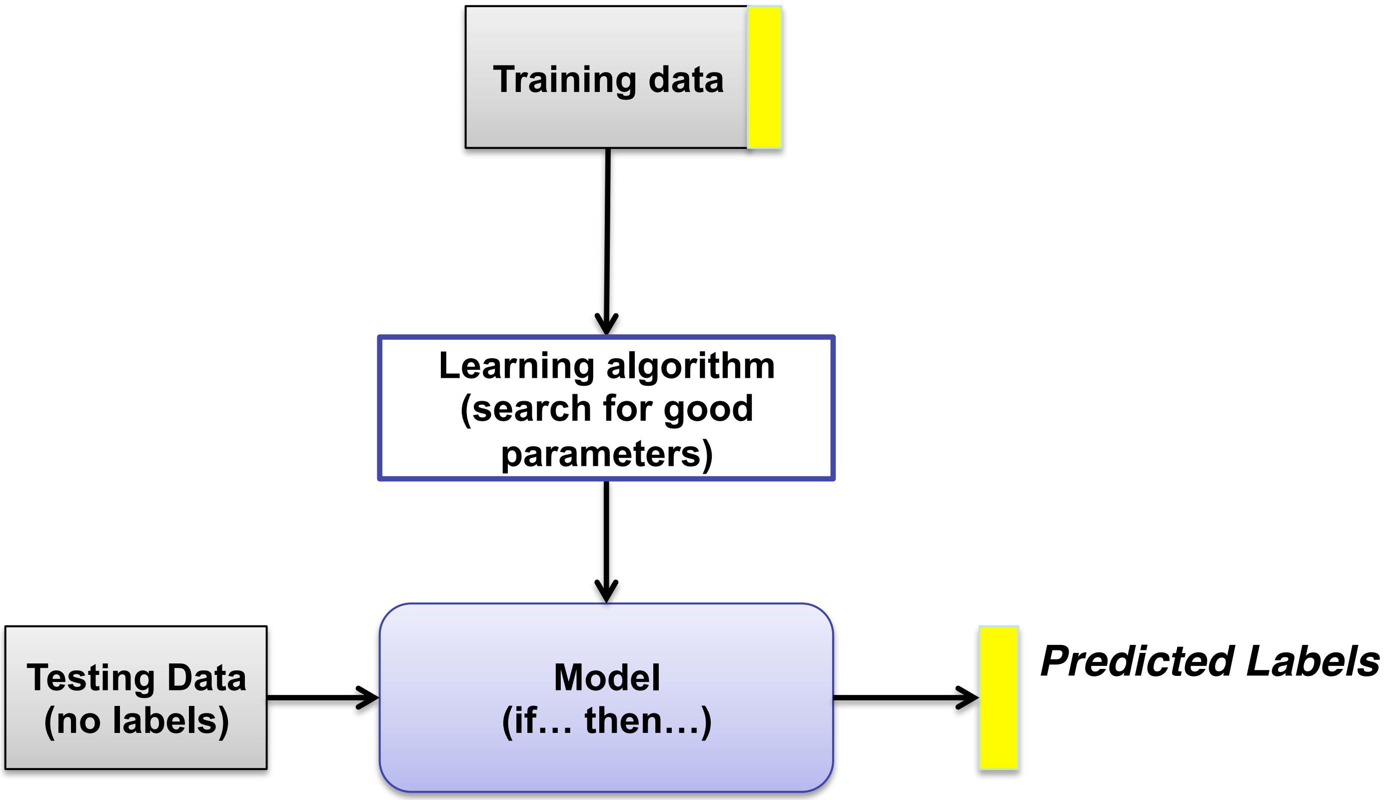
Learning algorithm for the Perceptron

```
initialise weights to random numbers in range -1 to +1
for n = 1 to NUM_ITERATIONS
        for each training example (x,y)
                calculate activation
                for each weight
                        update weight by learning rule
                end
        end
end
```

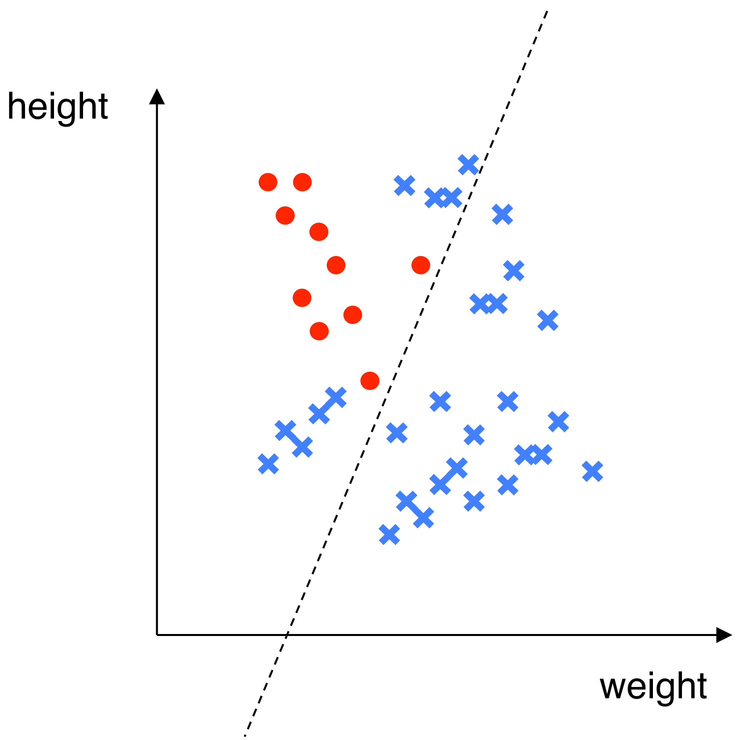
Perceptron convergence theorem:

If the data is linearly separable, then application of the Perceptron learning rule will find a separating decision boundary, within a finite number of iterations

Supervised Learning Pipeline for Perceptron



New data…. *"non-linearly separable"*



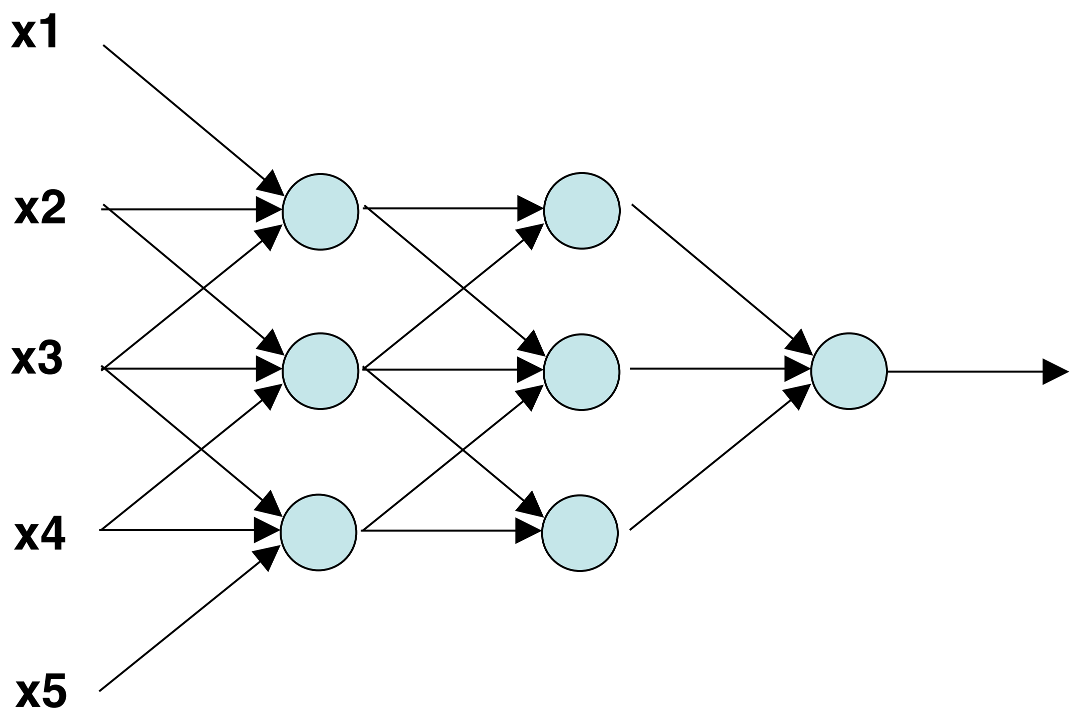
Our model does not match the problem!

(AGAIN!)

$$\text{if } \sum_{i=1}^{d} w_i x_i > t \text{ then "player" else "dancer"}$$

Many mistakes!

Multilayer Perceptron

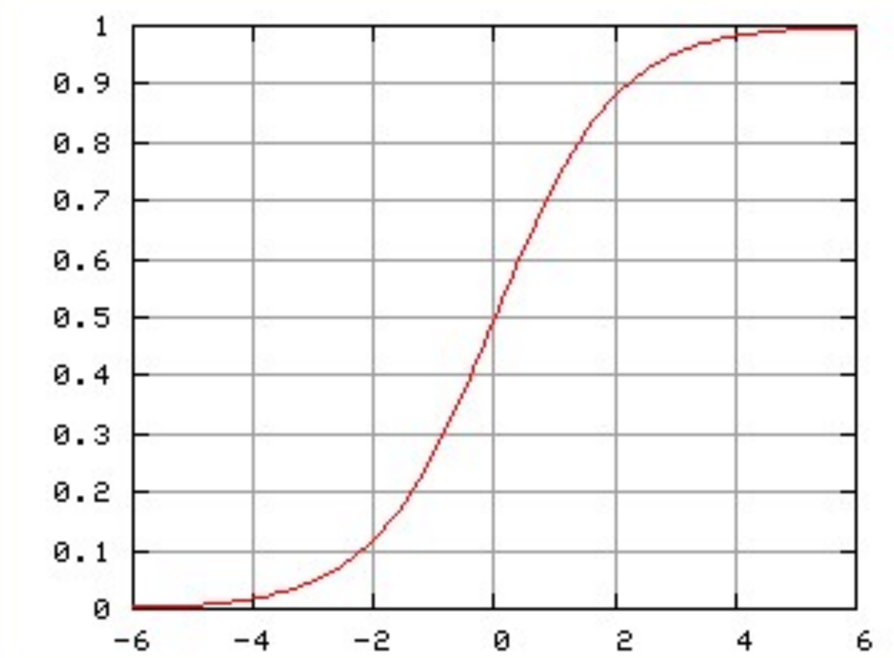


Sigmoid activation – no more thresholds needed ☺

$$\cancel{\text{if } \sum_{i=1}^{d} w_i x_i > t \text{ then } \hat{y} = 1 \text{ else } \hat{y} = 0}$$

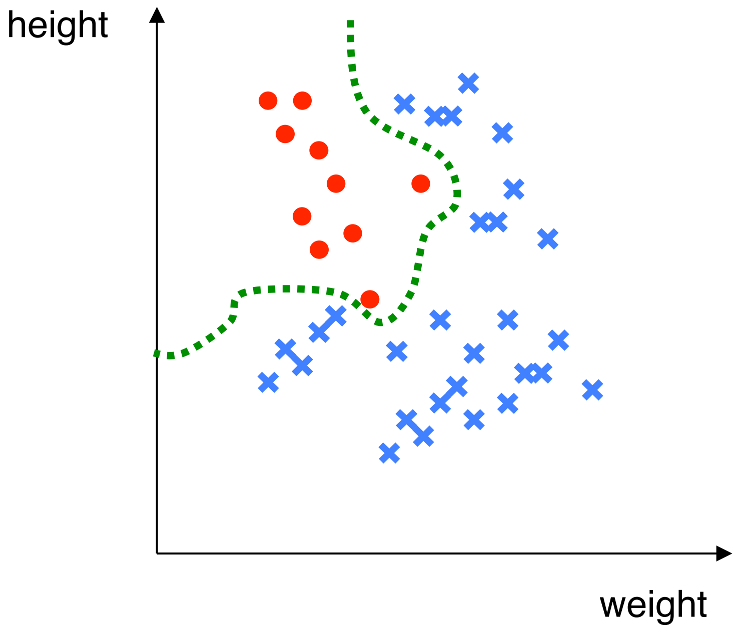
$$a = \frac{1}{1 + \exp(-\sum_{i=1}^{d} w_i x_i)}$$

activation level



$$\sum_{i=1}^{d} w_i x_i$$

MLP decision boundary – nonlinear problems, solved!



Neural Networks - summary

Perceptrons are a (simple) emulation of a neuron.

Layering perceptrons gives you… a multilayer perceptron.
An MLP is <u>one</u> type of neural network – there are others.

An MLP with sigmoid activation functions can solve highly nonlinear problems.

Downside – we cannot use the simple perceptron learning algorithm.

Instead we have the "backpropagation" algorithm.

This is outside the scope of this introductory course.