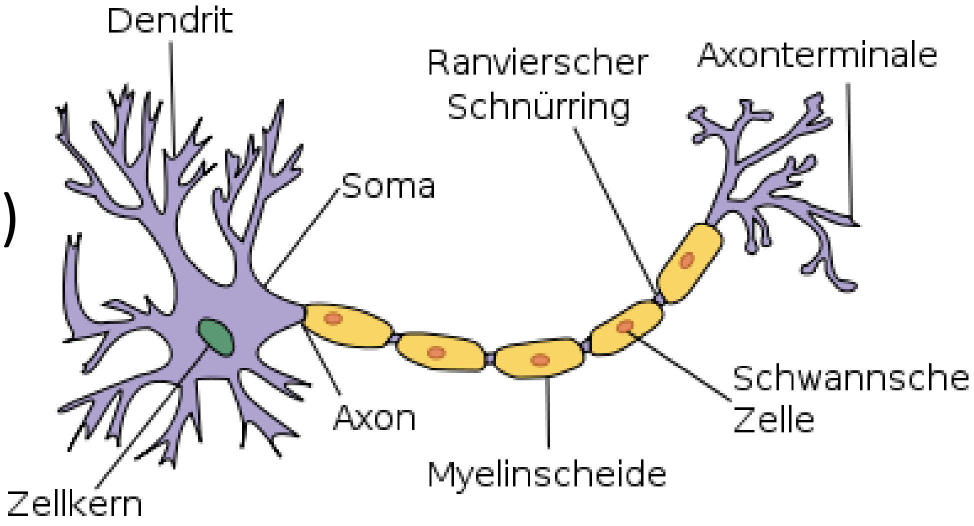
Machine Learning

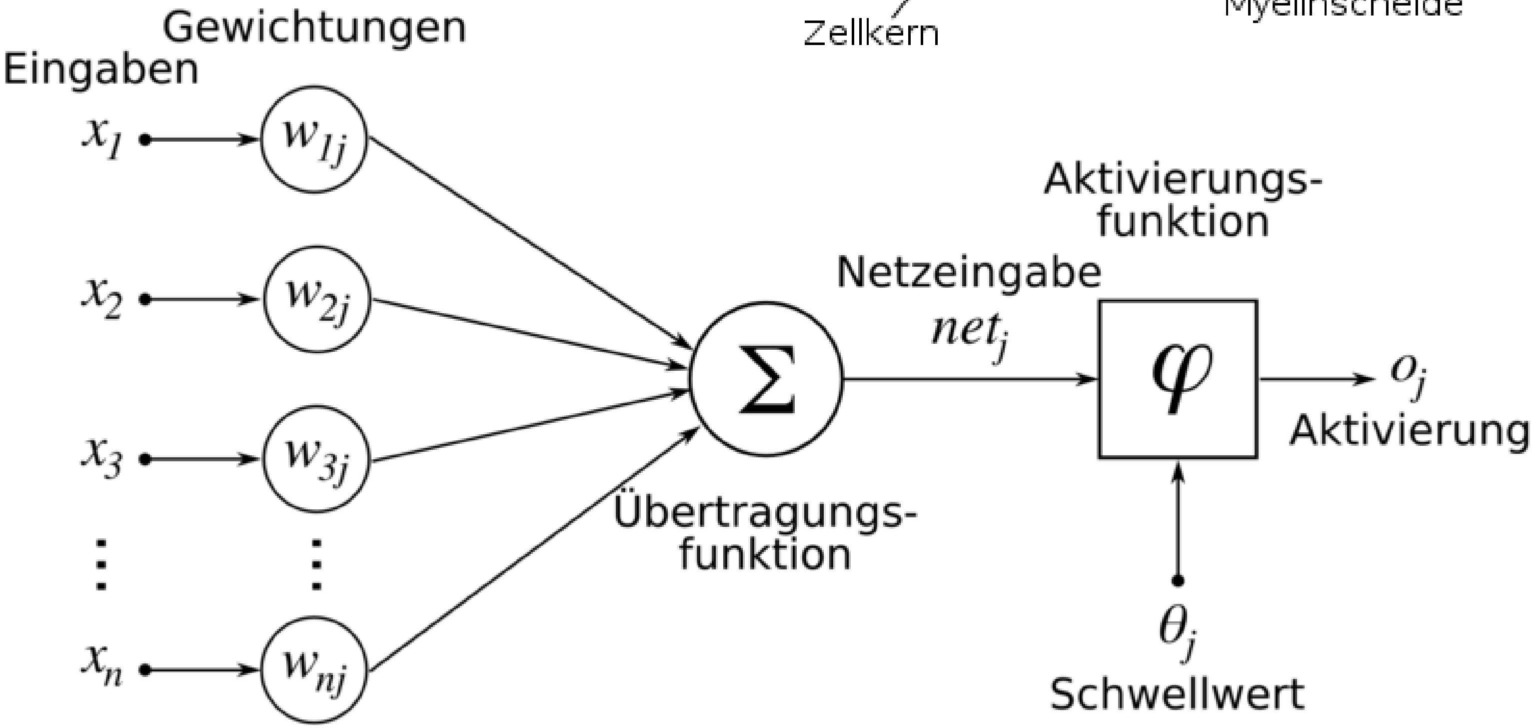
Neuron

28/11/2013

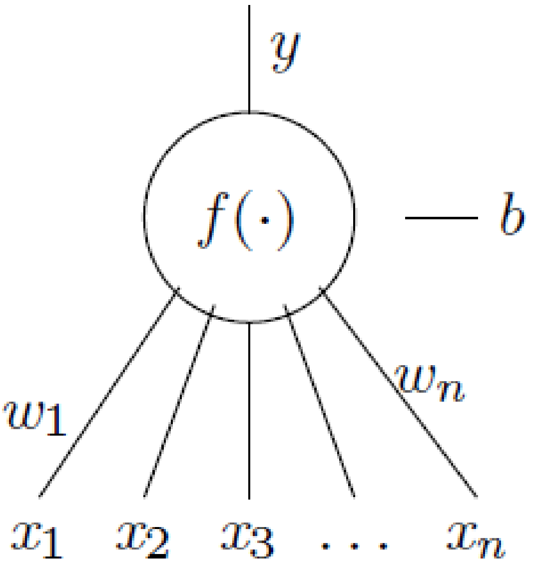
Neuron

Hunan vs. Computer
(two nice pictures from Wikipedia)





Neuron (McCulloch and Pitt, 1943)



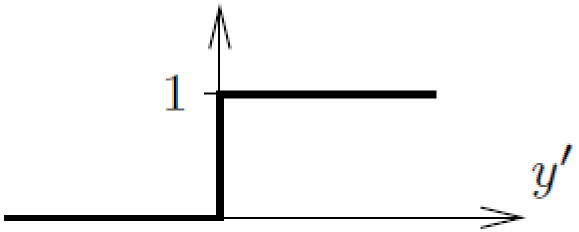
$$\langle x, w \rangle \lessgtr b$$

Input: $x \in \mathbb{R}^n$

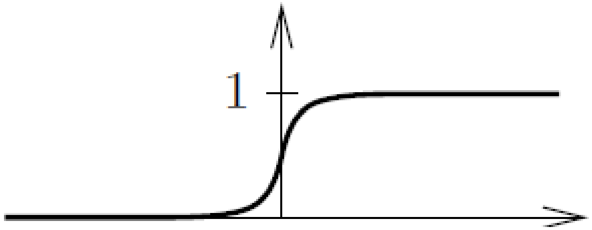
Weights: $w \in \mathbb{R}^n$

Activation: $b \in \mathbb{R}$

Output: $y = f(y' - b) = f(\langle w, x \rangle - b)$



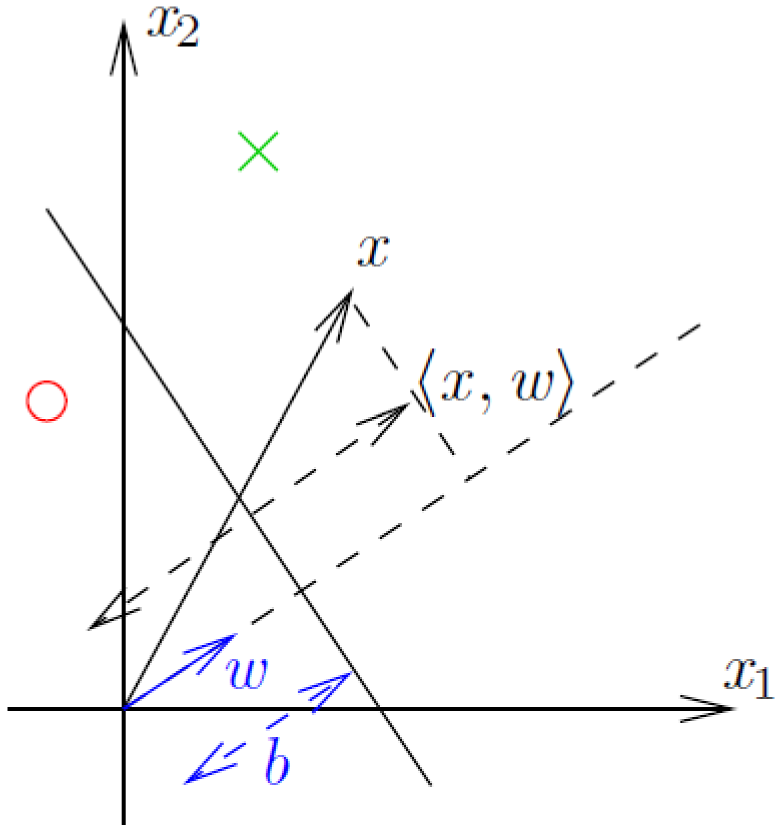
Step-function $f(y') = \begin{cases} 1 & \text{If } y' > 0 \\ 0 & \text{otherwise} \end{cases}$



Sigmoid-function (differentiable!!!)

$$f(y') = \frac{1}{1+\exp(-y')}$$

Geometric interpretation



$$\langle x, w \rangle = \|x\| \cdot \|w\| \cdot \cos\phi$$

Let w be normalized, i.e. $\|w\| = 1$

$\Rightarrow \|x\| \cdot \cos\phi$ the length of the projection of x onto w .

Separation plane: $\langle x, w \rangle = const$

Neuron implements a linear classifier

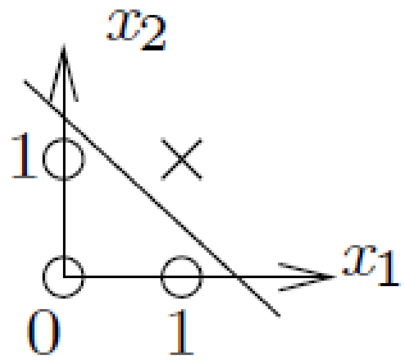
Special case – Boolean functions

Input: $x = (x_1, x_2),\ x_i \in \{0, 1\}$

Output: $y = x_1 \& x_2$

Find w and b so, that $\mathrm{step}(w_1 x_1 + w_2 x_2 - b) = x_1 \& x_2$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$w_1 = w_2 = 1,\ b = 1.5$$

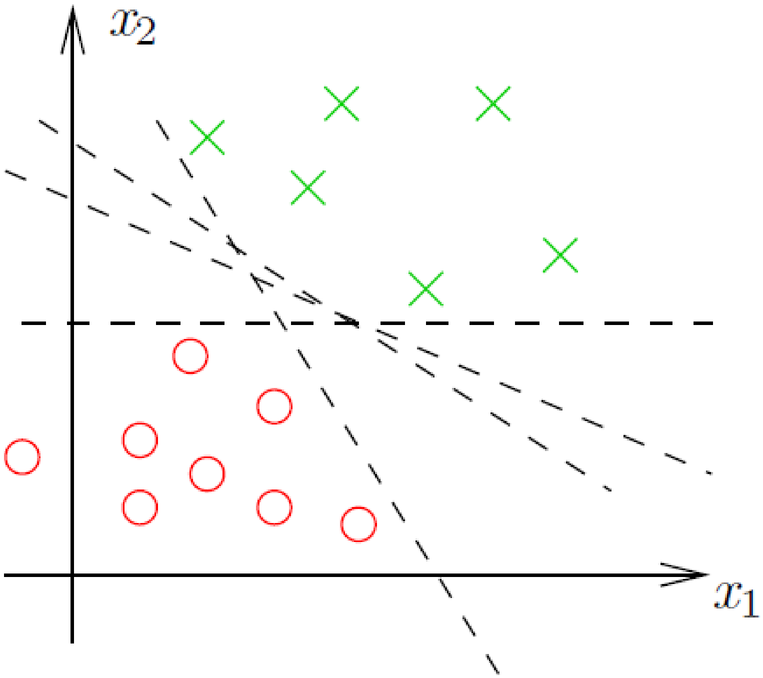
Disjunction, other Boolean functions, but XOR

Learning

Given: training data $\big((x^1,y^1),(x^2,y^2),\dots,(x^L,y^L)\big),\ x^l\in\mathbb{R}^n,\ y^l\in\{0,1\}$

Find: $w\in\mathbb{R}^n,\ b\in\mathbb{R}$ so that $f(\langle x^l,w\rangle-b)=y^l$ for all $l=1,\dots,L$

For a step-neuron: system of linear inequalities



$$\begin{cases} \langle x^l,w\rangle > b & \text{if} \quad y^l=1,\\ \langle x^l,w\rangle < b & \text{if} \quad y^l=0. \end{cases}$$

Solution is not unique in general !

“Preparation 1”

Eliminate the bias:

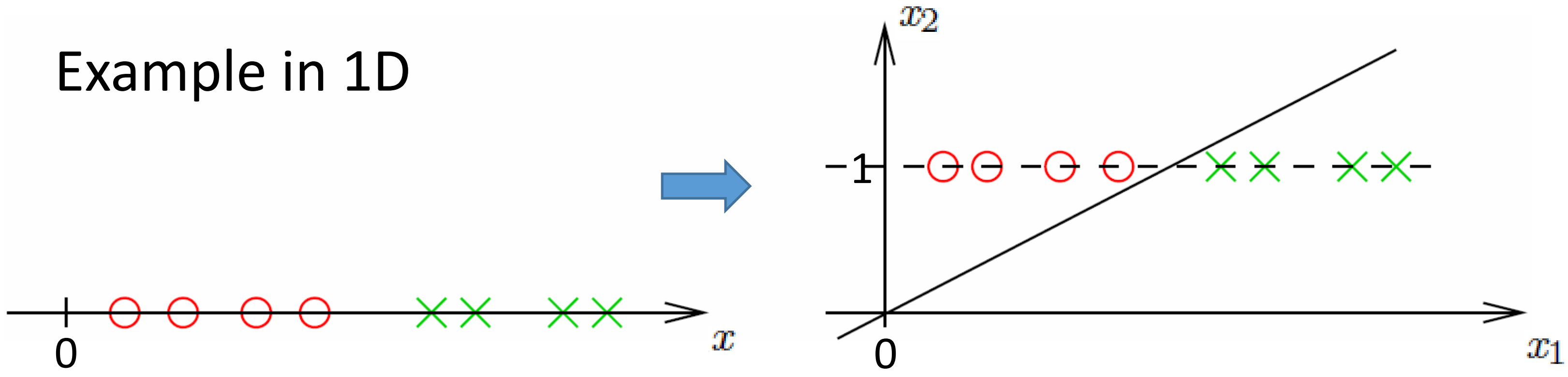
The trick – modify the training data

$$x = (x_1, x_2, \ldots, x_n) \Rightarrow \tilde{x} = (x_1, x_2, \ldots, x_n, 1)$$

$$w = (w_1, w_2, \ldots, w_n) \Rightarrow \tilde{w} = (w_1, w_2, \ldots, w_n, -b)$$

$$\langle x^l, w\rangle \gtrless b \Rightarrow \langle \tilde{x}^l, \tilde{w}\rangle \gtrless 0$$

Example in 1D



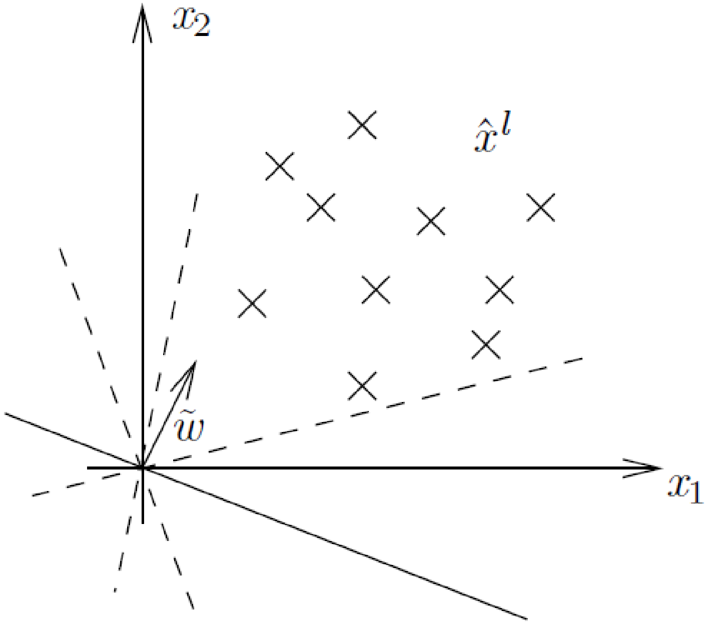
non-separable without the bias

separable without the bias

"Preparation 2"

Remove the sign:

The trick – the same



$\hat{x}^l = \tilde{x}^l$ for all with $y^l = 1$

$\hat{x}^l = -\tilde{x}^l$ for all with $y^l = 0$

All in all:

$$\begin{cases} \langle x^l, w \rangle > b & \text{if} \quad y^l = 1 \\ \langle x^l, w \rangle < b & \text{if} \quad y^l = 0 \end{cases} \quad \Rightarrow \quad \boxed{\langle \hat{x}^l, \tilde{w} \rangle > 0 \quad \forall l}$$

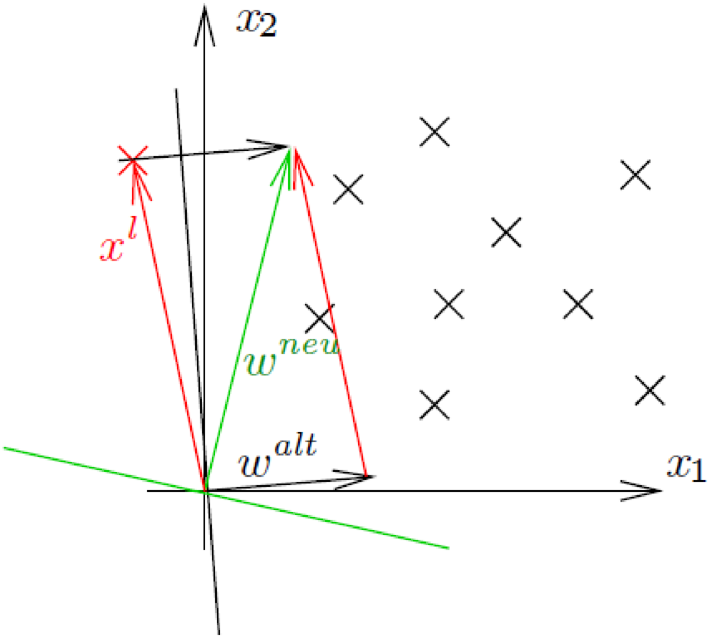
Perceptron Algorithm (Rosenblatt, 1958)

Solution of a system of linear inequalities:

1. Search for an equation that is not satisfied, i.e.

$$\langle x^l, w \rangle \leq 0$$

2. If not found – Stop
   else update $w^{neu} = w^{alt} + x^l$
   go to 1.



- The algorithm terminates if a solution exists (the training data are separable)
- The solution is a convex combination of the data points

Proof of convergence

The idea: look for quantities that

a) grow/decrease quite fast,

b) are bounded.

Consider the length of $w^{(n)}$ at n-th iteration:

$$||w^{(n+1)}||^2 = ||w^{(n)} + x^i||^2 = ||w^{(n)}||^2 + 2\langle w^{(n)}, x^i\rangle + ||x^i||^2 \leq ||w^{(n)}||^2 + D^2$$

with $D = \max_l ||x^l||$

<0, because added by the algorithm

$$\boxed{||w^{(n)}|| \leq \sqrt{n} D}$$

Proof of convergence

Another quantity – the projection of $w^{(n)}$ onto the **solution** w^*.

$$\frac{\langle w^{(n+1)}, w^*\rangle}{||w^*||} = \frac{\langle w^{(n)}, w^*\rangle}{||w^*||} + \frac{\langle x^i, w^*\rangle}{||w^*||} \geq \frac{\langle w^{(n)}, w^*\rangle}{||w^*||} + \epsilon$$

>0, because of the solution

With $\epsilon = \min_l \langle x^l, w^*\rangle / ||w^*||$ – the **Margin**

$$\boxed{\frac{\langle w^{(n)}, w^*\rangle}{||w^*||} \geq n\epsilon}$$

Proof of convergence

All together:

$$\boxed{||w^{(n)}|| \leq \sqrt{n}D} \text{ and } \boxed{\frac{\langle w^{(n)}, w^{*}\rangle}{||w^{*}||} \geq n\epsilon} \Rightarrow \frac{\langle w^{(n)}, w^{*}\rangle}{||w^{*}|| \cdot ||w^{(n)}||} \geq \sqrt{n}\frac{\epsilon}{D}$$

But $1 \geq \dfrac{\langle w^{(n)}, w^{*}\rangle}{||w^{*}|| \cdot ||w^{(n)}||}$ (Cauchy-Schwarz inequality)

So $1 \geq \sqrt{n}\dfrac{\epsilon}{D}$ and finally $n \leq \dfrac{D^2}{\epsilon^2}$

If the solution exists,

the algorithm converges after D^2/ϵ^2 steps at most.

An example problem

Consider another decision rule for a real valued feature $x \in \mathbb{R}$:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = \sum_i a_i x^i \gtrless 0$$

It is not a linear classifier anymore but a polynomial one.

The task is again to learn the unknown coefficients a_i
given the training data $\big((x^l, y^l) \ldots\big)$, $x^l \in \mathbb{R}$, $y^l \in \{0, 1\}$

Is it also possible to do that in a “Perceptron-like” fashion ?

An example problem

The idea: reduce the given problem to the Perceptron-task.

Observation: although the decision rule is not linear with respect to x, it is still linear with respect to the **unknown** coefficients a_i

The same trick again – modify the data:

$$\begin{aligned} w &= (a_n, a_{n-1}, \dots, a_1, a_0) \\ \tilde{x} &= (x^n, x^{n-1}, \dots, x, 1) \end{aligned} \quad \Rightarrow \quad \sum_i a_i x^i = \langle \tilde{x}, w \rangle$$

In general, it is very often possible to learn non-linear decision rules by the Perceptron algorithm using an appropriate transformation of the input space (further extension – SVM).

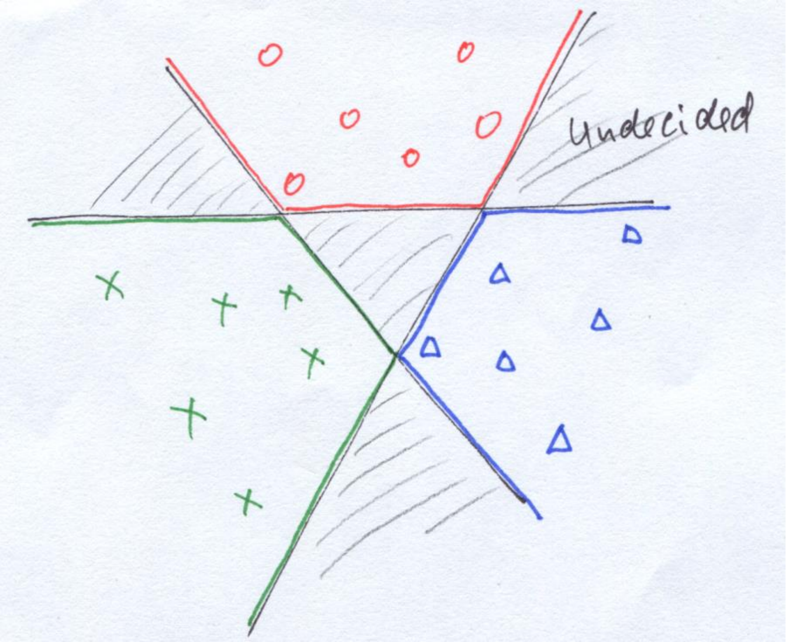
Many classes

Before: two classes – a mapping $\mathbb{R}^n \rightarrow \{0,1\}$

Now: many classes – a mapping $\mathbb{R}^n \rightarrow \{1 \ldots K\}$

How to generalize ? How to learn ?

Two simple (straightforward) approaches:

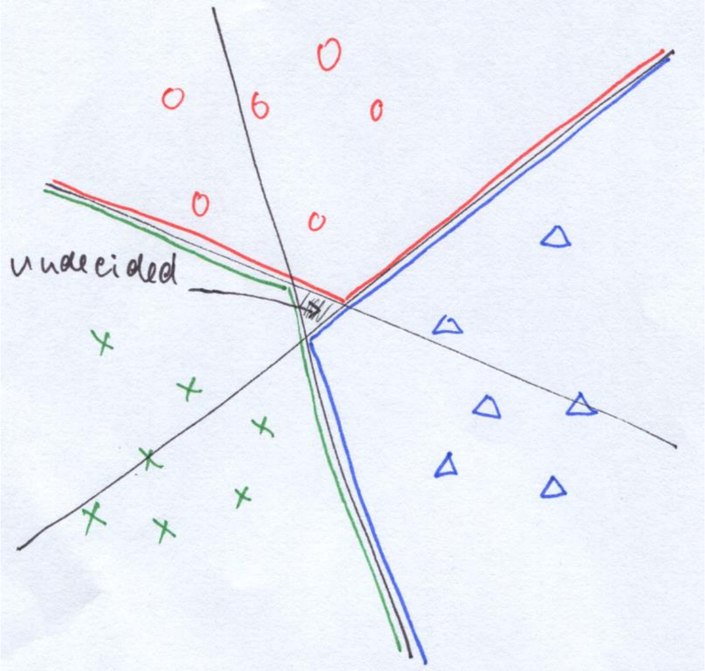


The first one: "one vs. all" – there is one binary classifier per class, that separates this class from all others.

The classification is ambiguous in some areas.

Many classes

Another one:
“pairwise classifiers” – there is a classifier for each class pair



Less ambiguous, better separable.

However:

$K(K-1)/2$ binary classifiers instead of K in the previous case.

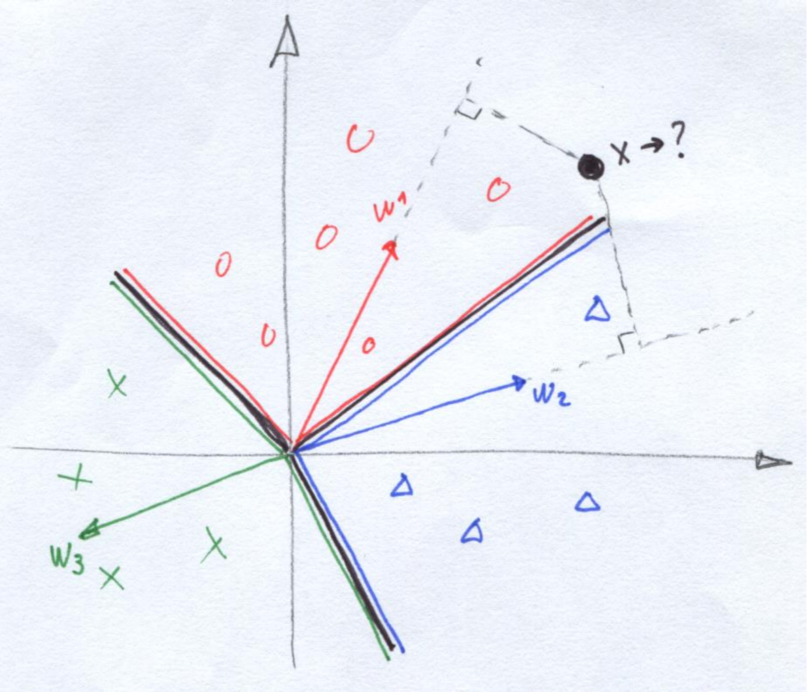
The goal:

- no ambiguities,
- K parameter vectors

Fisher Classifier

Idea: in the binary case the output y is the more likely to be "1" the greater is the scalar product $\langle x, w \rangle$ → generalization:

$$y = \arg\max_k \langle x, w_k \rangle$$



Geometric interpretation (let w_k be normalized)

Consider projections of an input vector x onto vectors w_k

The input space is partitioned into the set of convex cones.

Fisher Classifier

Given: training set $\big((x^1, k^1) \ldots (x^l, k^l)\big),\ x^l \in \mathbb{R}^n,\ k^l \in \mathbb{R}$

To be learned: weighting vectors

The task is to choose w_k so that

$$y^l = \arg\max_k \langle x^l, w_k \rangle \quad \forall l$$

It can be equivalently written as

$$\langle x^l, w_{y^l} \rangle > \langle x^l, w_k \rangle \quad \forall l, k \neq y^l$$

– a system of linear inequalities, but a “heterogenic” one.

The trick – transformation of the input/parameter space.

Fisher Classifier

Example for three classes: Consider e.g. a training example $(x, 1)$, it leads to the following inequalities:

$$\langle x, w_1 \rangle > \langle x, w_2 \rangle$$
$$\langle x, w_1 \rangle > \langle x, w_3 \rangle$$

Let us define the new parameter vector as

$$\tilde{w} = (w_{11}, \ldots w_{1n}, w_{21}, \ldots w_{2n}, w_{31}, \ldots w_{3n})$$

i.e. we "concatenate" all w_k to a single vector.

For each inequality (see example above) we introduce a "data point":

$$\tilde{x} = (x_1, \ldots, x_n, -x_1, \ldots, -x_n, \quad 0, \ldots, \quad 0)$$
$$\tilde{x} = (x_1, \ldots, x_n, \quad 0, \ldots, \quad 0, -x_1, \ldots, -x_n)$$

→ all inequalities are written in form of a scalar product $\langle \tilde{x}, \tilde{w} \rangle > 0$

Solution by the Perceptron Algorithm.

Conclusion

Today:

- Neuron – linear classifier
- Perceptron Algorithm – simple update rule, convergence
- Fisher classifier – „Multiclass Perceptron“

Next Lecture – Neuronal networks:

- Feed-Forward networks
- Nopfield networks
- Clustering, Cohonen networks