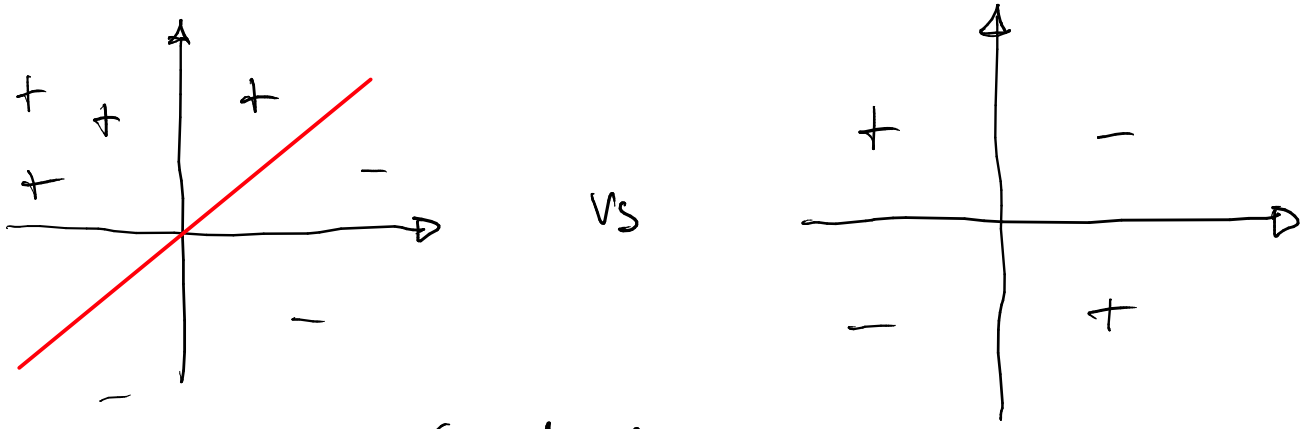
7. Linear model for classification

Learning a function $f: X \to Y$ with $X \subseteq \mathbb{R}^d$ and $Y = \{c_1, \dots, c_K\}$. The idea of these methods is to consider the ML problem as an optimization problem and find the solution to this one.

We assume that data are linearly separable, this is a particular feature of the data:

> INSTANCES are LINEARLY SEPARABLE iff $\exists$ a hyperplane that divide the instance space into two regions such that differently classified instances are separated



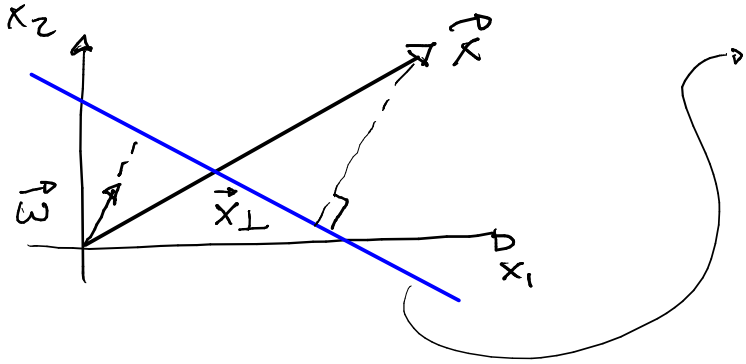
Graphical representation of the dataset $X \subseteq \mathbb{R}^2$ and $Y = \{+, -\}$

If you have more than two classes the model is described by K - different functions, the multi-class function is: $y_k(\vec{x}) \vec{w}_k^T \vec{x} + w_{k0}$

In two class we need only one function:

$y(\vec{x}) = \vec{w}^T \vec{x} + w_0 = \tilde{\vec{w}}^T \tilde{\vec{x}}$ with $\tilde{\vec{w}} = \begin{pmatrix} w_0 \\ \vec{w} \end{pmatrix}$, $\tilde{\vec{x}} = \begin{pmatrix} 1 \\ \vec{x} \end{pmatrix}$

Let's see the geometrical interpretation of those linear functions, starting by the binary classification problem:



line in which $y(\vec{x}) = 0$

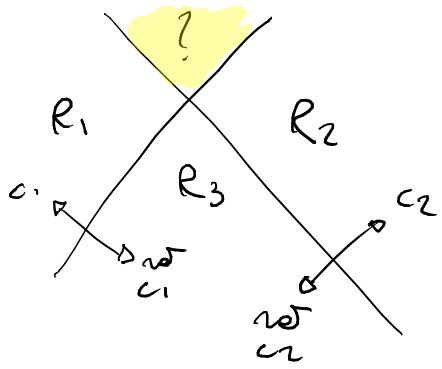
$\vec{w}$ is $\perp$ to this line.

This line is partitioning the space and on the sides you have different points depending on $y(\vec{x})$ $\begin{cases} \geq 0 \\ < 0 \end{cases}$

We want to find the best line since it will be used to classify.

When we have k-classes, with $k \geq 2$ we cannot use simple tricks

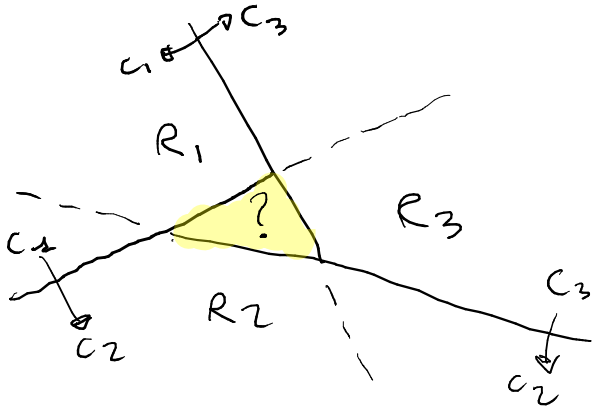
1-Vs-REST



You use binary classifier $(K-1)$ in general. The problem is that there is a region in which we cannot determine the right class.

1-Vs-1

Quadratic number of classifiers $k(k-1)/2$ and you classify by considering each possible pair



We still have a region of uncertainty.

We cannot transform k-class classifier into a set of binary classifiers.

We have to change the model:

$y_k(\vec{x}) = \vec{w}_k^T \vec{x} + w_{k0}$ compute the $\vec{w}$

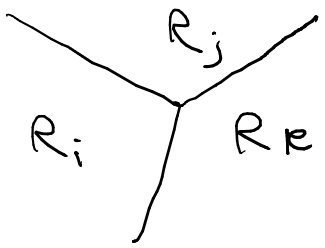
Then assign to a new instance $\vec{x}$ class C_k if:

$$\boxed{y_k(\vec{x}) > y_j(\vec{x}) \quad \forall j \neq k}$$

Given this formulation we can derive the equation of the hyperplane separating the two classes C_k and C_j:

$$(\vec{w}_k - \vec{w}_j)^T \vec{x} + (w_{0k} - w_{0j}) = 0$$

And the representation is like this:



This separates correctly the regions: the output of the model is a partition of the space. If you don't model the problem like this you have troubles!

Compact notation:

$y_k(\vec{x}) = \vec{w}_k^T \vec{x} + w_{0k}$ $\vec{x}$ not in dataset

$$y(x) = \begin{pmatrix} y_1(\vec{x}) \\ \vdots \\ y_K(\vec{x}) \end{pmatrix} = \begin{pmatrix} \tilde{\vec{w}}_1^T \tilde{\vec{x}} \\ \vdots \\ \tilde{\vec{w}}_K^T \tilde{\vec{x}} \end{pmatrix} = \tilde{\vec{W}}^T \tilde{x}$$

$$\tilde{\vec{w}}_k = \begin{pmatrix} w_{0k} \\ \vec{w}_k \end{pmatrix} \qquad \tilde{\vec{x}} = \begin{pmatrix} 1 \\ \vec{x} \end{pmatrix} \leftarrow \text{dummy component to get } w_{0k}$$

Given a multi-class classification problem and a dataset D with linearly separable data, determine $\tilde{W}$ such that $\vec{y}(\vec{x}) = \tilde{\vec{W}}^T \tilde{\vec{x}}$ is the k-class discriminant

LEAST SQUARES

We have a dataset $\mathcal{D}$ with inputs and outputs, $\mathcal{D} = \{(\vec{x}_n, \vec{t}_n)_{n=1}^N\}$ and we use a particular convention to represent $\vec{t}$, in particular we use the 1-of-K coding scheme (only one component is 1):

$$x \in C_k \rightarrow t_k = 1, \; t_j = 0 \;\; \forall j \neq k, \; \vec{t}_n = (0 \dots 1 \dots 0)^T$$

So now we can represent our dataset in a matrix form:

$$\tilde{X} = \begin{pmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{pmatrix} \qquad T = \begin{pmatrix} \vec{t}_1^T \\ \vdots \\ \vec{t}_N^T \end{pmatrix}$$

$\tilde{X}$: design matrix

We define now an error function, where the error is the difference from the predictions and the value that is in the dataset.

We consider the LEAST SQUARES ERROR FUNCTION:

$$E(\tilde{W}) = \frac{1}{2} Tr\left\{ (\tilde{X}\tilde{W} - T)^T (\tilde{X}\tilde{W} - T) \right\}$$

$\tilde{X}\tilde{W}$: prediction on the input; T: real value of the input

$E(\tilde{W}) = 0$ when $\tilde{W}$ predicts the same value of T for each value in the dataset (for each sample)

TASK: We want to find the value of $\tilde{W}$ that minimize the error. This is a quadratic problem, it can be solved with a close form solution:

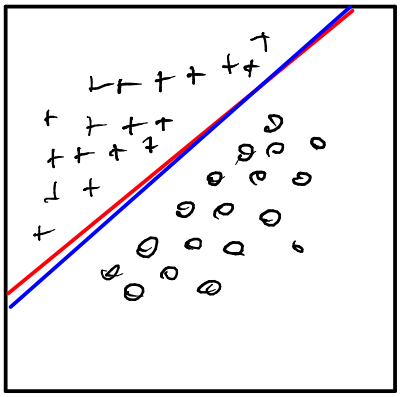
$$\tilde{W} = \underbrace{(\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T}_{\tilde{X}^\#} T$$

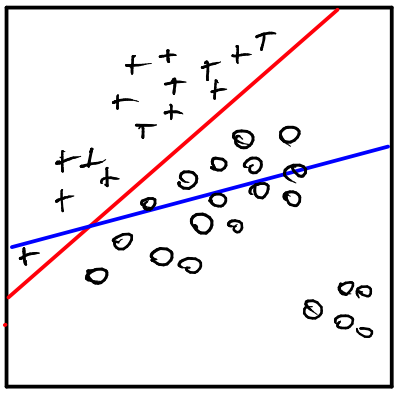
$\tilde{X}^\#$ pseudoinverse of X

$\vec{y}(\vec{x}^*) = \vec{T}^T (\tilde{X}^*)^T \tilde{X}$ → simple and easy.

INCONVENIENTS: Works very well for some dataset but is NOT ROBUST TO OUTLIERS.
(outliers = points not coming from the same distribution)

Least square considers a way of minimizing the distance from the computed line.



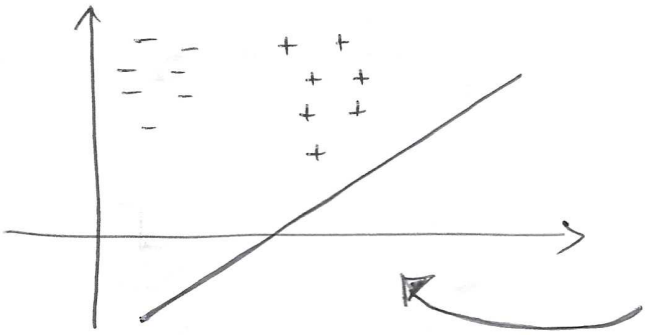


Outliers afraid the line.

Fischer's linear discriminant

For this problem we consider only 2 classes and a bidimensional space. We want to determine $y = \vec{w}^T \vec{x}$ and classify $\vec{x} \in C_1$ if $y \geq -w_0$, $\vec{x} \in C_2$ if $y < -w_0$.

The geometry of $\vec{w}^T \vec{x}$ is just a projection of $\vec{x}$ on the line determined by $\vec{w}$. One way to compute the separation line is the following!



We can consider all projection of the points on a line. According to different lines the projection is diff. IF THE LINE IS NOT "WELL-ORIENTED" THE PROJECTIONS OVERLAP !

We look for a line that MINIMIZE the overlapping or better that MAXIMIZE THE SEPARATION of the points ! You can consider the center of mass for each class (Centroids / means):

$$\vec{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \vec{x}_n \qquad \vec{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \vec{x}_n$$

CHOOSE $\vec{w}$ that maximizes $J(\vec{w}) = \vec{w}^T (\vec{m}_2 - \vec{m}_1)$ s.t. $\|\vec{w}\| = 1$. You project only the mean and you can do it for all possible function and looking for the one that maximizes separation.

GOOD! BUT STILL NOT PERFECT because you consider only the means, YOU DO NOT CONSIDER THE COVARIANCE. If we consider the covariance you can rotate the line in such a way it aligns with covariances. COVARIANCE ALLOW THE MODEL TO ROTATE!

$$J(\vec{w}) = \frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$$

Fischer criterion.

$$S_B = (\vec{m}_2 - \vec{m}_1)(\vec{m}_2 - \vec{m}_1)^T$$

Between classes

$$S_W = \sum_{n \in C_1} (\vec{x}_n - \vec{m}_1)^T (\vec{x}_n - \vec{m}_1)$$

within ... Σ (same for C_2)

$$\vec{w}^* \propto S_W^{-1}\left(\vec{m}_2 - \vec{m}_1\right)$$

$$J(\vec{w}) = \left\{\left(W^T S_W W\right)^{-1}\left(W^T S_B W\right)\right\} \text{ with } k \geq 2 \text{ classes.}$$

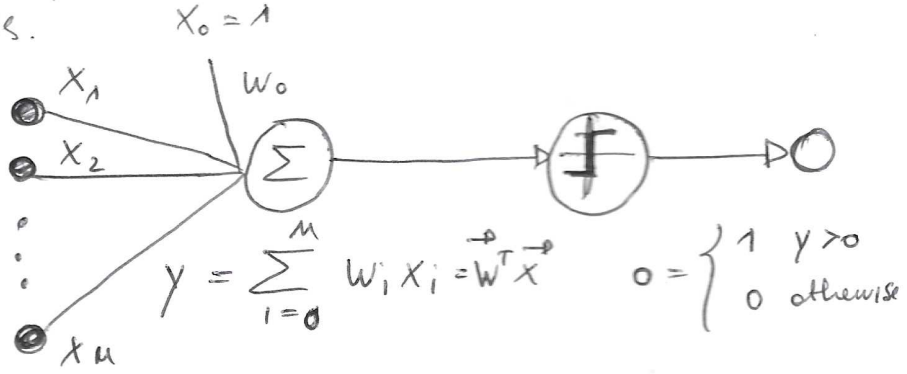
BETTER THAN LEAST SQUARES, more robust to outliers and better with more than two classes.

PERCEPTRON

↓

BASIC UNIT FOR A NEURAL NETWORK.

No close form, it will use an iterative method. We still define an optimization problem, but we solve it with an iterative process.



The model is defined as a linear combination of the input dimensions with some weights. We have another portion in which we apply the sign function, that is discontinuos.

OUR GOAL IS TO FIND w_i BY USING AN ITERATIVE PROCESS, At some in the execution of the algorithm you receive new information, in particular new samples from dataset, and you want to update w_i:

TRAINING RULE: $w_i \leftarrow w_i + \Delta w_i$, $\Delta w_i = \eta\left(t - o\right) x_i$

(t: value in D; o: prediction)

The bigger is the error the bigger is the update.

Linear

Unthreshold hunit!

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{m=1}^{N}\left(t_m - o_m\right)^2 = \frac{1}{2}\sum_{m=1}^{N}\left(t_m - \vec{w}^T \vec{x}_m\right)^2$$

Learn w_i to minimize the squared error.

To minimize we can use a gradient method, computing the derivative with respect to each component of $\vec{w}$!

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \vec{w}^T x_n \right)^2 = \sum_{n=1}^{N} \left(t_n - \vec{w}^T x_n \right)(-x_{i,n})$$

i, n.

$$\Delta w_i = (-) \eta \frac{\partial E}{\partial w_i} = -\eta \sum_{n=1}^{N} \left(t_n - \vec{w}^T x_n \right)(-x_{i,n})$$

η IS A SMALL CONSTANT CALLED "LEARNING RATE". We have a minus because we want to go in the negative direction of the gradient and we want to minimize the error. We can apply the algorithm in:

BATCH MODE vs **INCREMENTAL MODE**

We compute the gradient $\forall$ samples in D

or

MINI-BATCH

use a subset $S \subset D$

correctly compute the gradient.

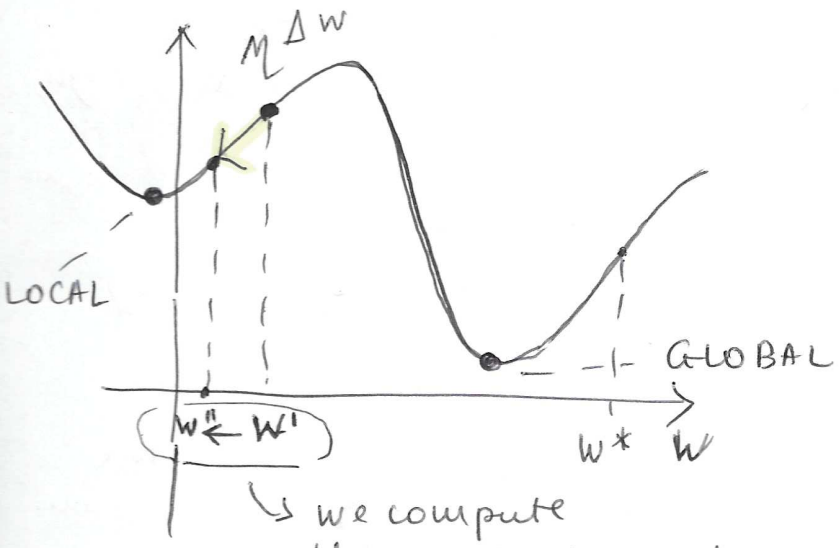
We consider only one sample at the time.

(✓) Better in ML because it converges faster! Less sensitive to local minima.

TERMINATION CONDITIONS:

- Predefined number of iterations
- Threshold on changes on the loss function $E(\vec{w})$.

Let's consider the following plot with **w** with only one dimension.



↳ we compute the gradient and move the value of w

▭ → direction of negative gradient.

THIS PROCESS WILL BRING TO THE LOCAL MINIMUM, NOT TO A GLOBAL

The solution of the iteration process depends on where you start. IF you start in $w = w'$ you will get to a local minimum, if you start in $w = w^*$ you will go to a global one.

STEPS of PERCEPTRON:

1. Initialize $\hat{w}$ (small random values)
2. Repeat until termination:

$$\hat{w}_i \leftarrow \hat{w}_i + \Delta w_i$$

3. Output $\hat{w}$.

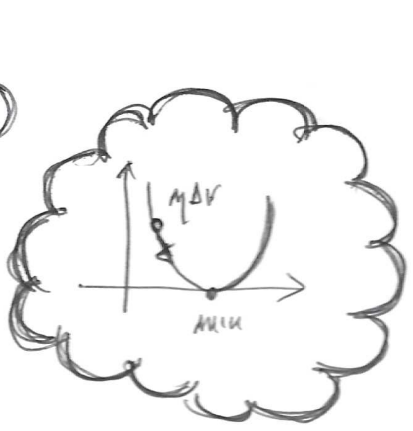
ANOTHER PROBLEM: The learning rate

→ if η is too small you will have too many steps

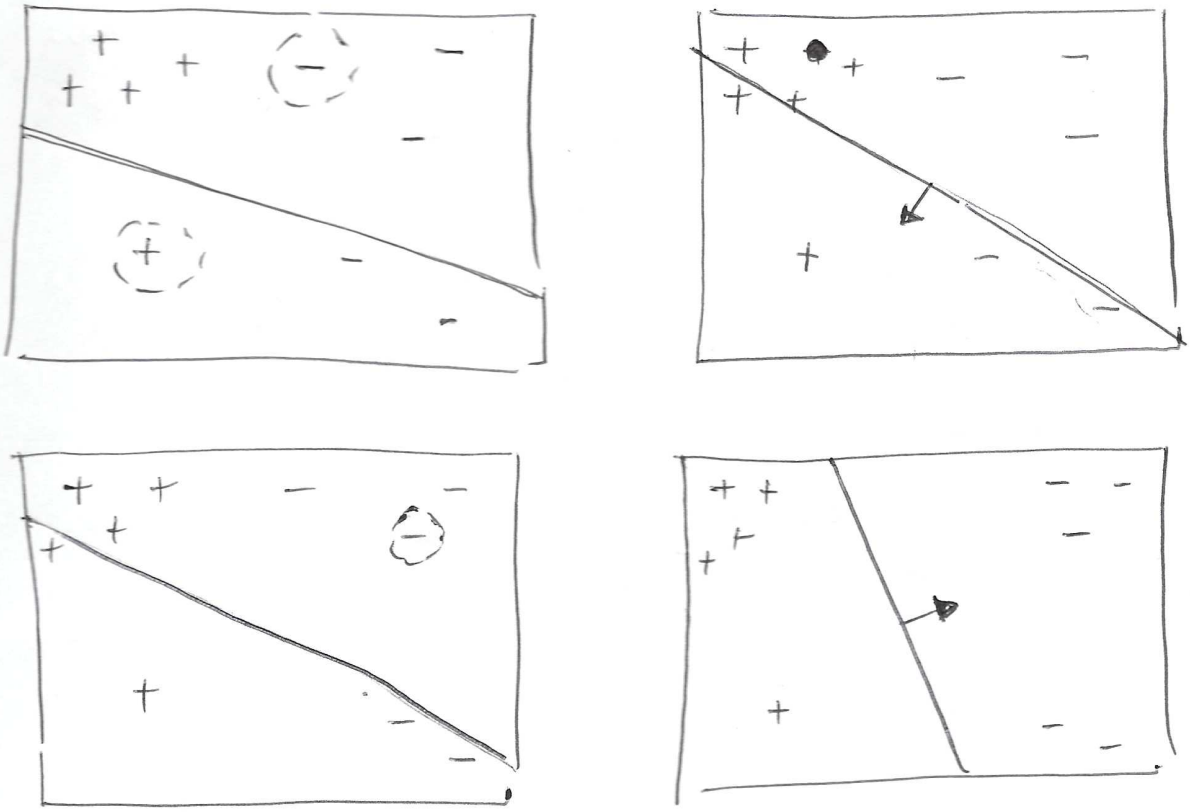
→ if η is too big you may have oscillations and you can diverge

Convergence condition:

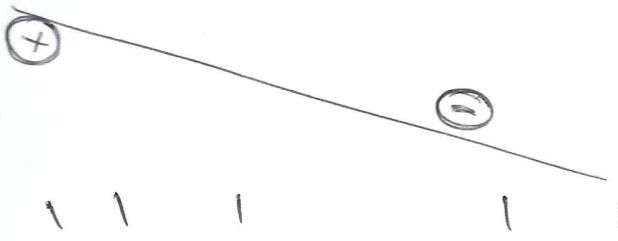
- data linearly separable
- η small enough (usually means slow convergence)



Plot the evolution over time of perceptron. Remember that every time we find one point for which the error is different the zero the parameter will change and the line is in some way attracted towards the data point. SINCE WE HAVE THE η TERM THE MOVEMENT IS LIMITED



The method stops when you find a line that correctly separates the dataset (the samples). This is true if η is small enough. IF η IS TOO SMALL YOU ALSO HAVE SOLUTION SENSITIVE TO NOISE, you will find a solution that has this shape



VERY RISKY ! A line is close to the circles points ! Usually this is not the line that separates the data.

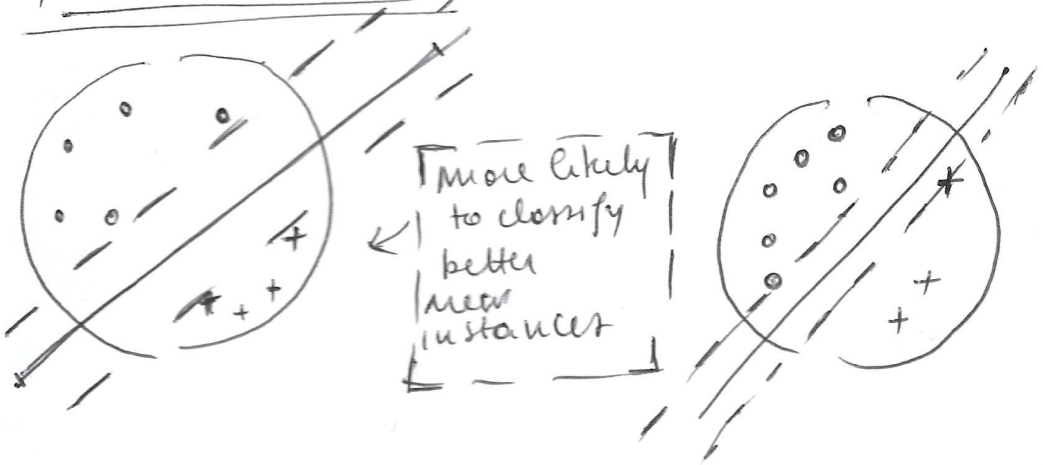
PERCEPTRON IS THE ONLY METHOD THAT IS NOT AFFECTED BY OUTLIERS (since it no considers means, variances, ...).

SUPPORT VECTOR MACHINES

MAIN GOAL : Find the direction that guarantees the MAXIMUM MARGIN between two categories.

Assumption: data linear separable.

The separation surface should be as far as possible from the other points (Probability of classify new instances is good, is what we want).



MAXIMUM MARGIN HYPERPLANE IS WHAT SVM IS SUPPOSED TO FIND.

Let's consider binary classification $y: X \to \{+1, -1\}$, with $D = \{(\vec{x}_M, t_M)_{M=1}^{N}\}$, $t_M \in \{-1, +1\}$ and a linear model:

$$\boxed{y(\vec{x}) = \vec{w}^T \vec{x} + w_0}$$

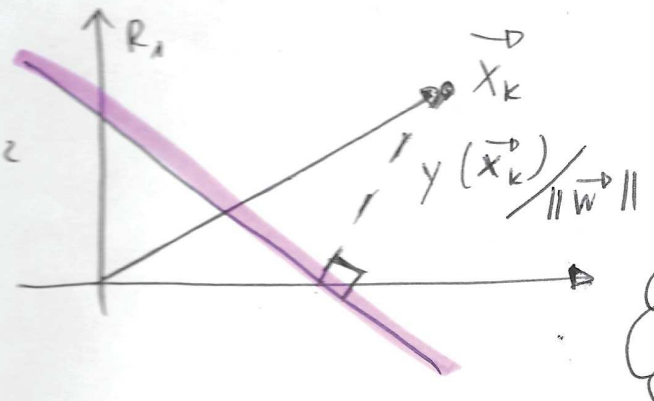
Assume D is linearly separable:

$$\exists\, \vec{w}, w_0 \text{ s.t. } \begin{cases} y(\vec{x}_M) > 0, & \text{if } t_M = +1 \\ y(\vec{x}_M) < 0, & \text{if } t_M = -1 \end{cases}$$

$$\Rightarrow \quad t_M\, y(x_M) > 0 \qquad \forall\, M = 1 \dots N$$

The margin is computed by considering the distance between the hyperplane and the closest point in the dataset.

The margin can be computed as $\frac{|y(\vec{x}_k)|}{\|\vec{w}\|}$.



The closest point is the one that minimizes the distance between the point itself and the hyperplane.

$$\min_{n=1,..,N} \frac{|y(\vec{x}_n)|}{\|\vec{w}\|} = \text{margin}$$

$$\min_{n=1...N} \frac{|y(\vec{x}_n)|}{\|\vec{w}\|} = \dots = \frac{1}{\|\vec{w}\|} \min_{n=1,..,N} [t_n(\vec{w}^T\vec{x}_n + w_0)]$$

$h = \vec{w}^T\vec{x} + w_0$ is the hyperplane. So, what I want? I WANT TO SELECT AMONG ALL POSSIBLE HYPERPLANES, THE ONE THAT MAXIMIZES THE MARGIN, aka the minimum distance between the hyperplane and any point in the dataset!

$h^* = \vec{w}^{*T}\vec{x} + w_0^*$ the hyperplane with maximum margin is computed as:

$$\vec{w}^*, w_0^* = \underset{\vec{w}, w_0}{\text{argmax}} \frac{1}{\|\vec{w}\|} \min_{n=1,..,N} [t_n(\vec{w}^T\vec{x} + w_0)]$$

We can rescale all the points in such a way we have: (WE CAN SCALE BY A CONSTANT FACTOR, does not affect the solution)

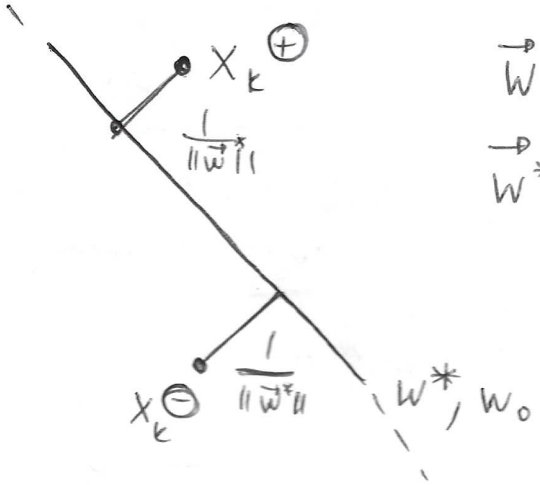
$$t_k(\vec{w}^T\vec{x}_k + w_0) = 1$$

The margin is 1, $\vec{x}_k$ is the closest point.

For all the other points the quantity is in general:

$$t_n(\vec{w}^T\vec{x}_n + w_0) \geq 1 \quad \forall n = 1, \dots, N$$

When we compute the optimal margin hyperplane we have at least 2 closest point (one for each side):



$$\vec{w}^{*T}\vec{x}_k^{\oplus} + w_0 = 1$$

$$\vec{w}^{*T}\vec{x}_k^{\ominus} + w_0 = -1$$

The canonical representation of the maximum margin hyperp. can be found by solving the optimization problem:

$$\max\left(\frac{1}{\|w\|}\right) = \min \frac{1}{2}\|w\|^2$$

useful since the problem is in a quadratic form that it is easy to solve

Subject to:

$$t_n\left(\vec{w}^T\vec{x}_n + w_0\right) \geq 1 \quad \forall_n$$

QP problem solved with Lagrangian method. The solution:

$$w^* = \sum_{n=1}^{N} a_n t_n \vec{x}_n$$

In this case the number of parameters is equal to N, that is the size of the dataset (a_i are the Lagrangian multipliers).

KARUSH KUHN TUCKER CONDITION:

for each $\vec{x}_n \in X_D$, either $a_n = 0$ or $t_n y(\vec{x}_n) = 1$

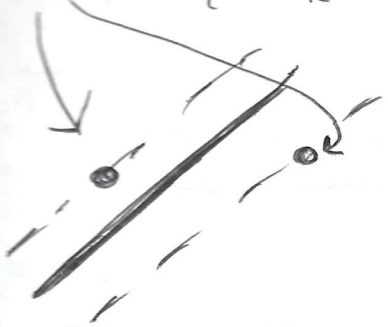
This means that many $a_n = 0$ and few of them are different from zero. $\vec{x}_n$ for which $a_n = 0$ do not contribute to the solution!

The solution is given only by those points (for which $a_M \neq 0$), and these points are very few, sometimes they are just two.

SUPPORT VECTORS:

Vectors for which the corresponding lagrangian multipliers $a_M \neq 0$ or in another words they are the point at distance 1 from the (maximum) margin hyperplane.

$$SV \equiv \{ \vec{x}_k \in X_D \mid t_k y(\vec{x}_k) = 1 \}$$



SO THE SOLUTION IS:

$$y(\vec{x}) = \underbrace{\sum_{\vec{x}_j \in SV} a_j t_j \vec{x}_j^T \vec{x}_j}_{\text{hyperplane expressed with SV}} + w_0 = 0$$

We can compute w_0 because we can exploit the property that $t_k y(\vec{x}_k) = 1$. Instead of using one particular SV a more stable solution is obtained by averaging over all the support vectors:

$$w_0 = \frac{1}{|SV|} \sum_{x_k \in SV} \left(t_k - \sum_{x_j \in SV} a_j t_j \vec{x}_j^T \vec{x}_j \right)$$

Note:

the original eq. for w_0 is obtained by multiply y for t_k:

$$t_k \left(\sum a_j t_j \vec{x}_j^T \vec{x}_I + w_0 \right) = 1$$ and remembering that $t_k^2 = 1$

REMEMBER THAT YOU HAVE AT LEAST TWO SV. The optimization problem for determining $\vec{w}$ (dimension $|x|$) has been transformed in an optimization problem for determing $\vec{a}$ (dimension $|D|$, most of them are zero)

EFFICIENT when $|X| < |D|$, very useful when $|X|$ is large or infinite.

What if data are "almost" linear separable?

→ Linear separable except for some points. Some methods do not work, SVM can be extended.

WE CAN TRASFORM THE OPT. PROBLEM for SVM BY introducing the slack variables!

ξ_n (one for each sample in D)

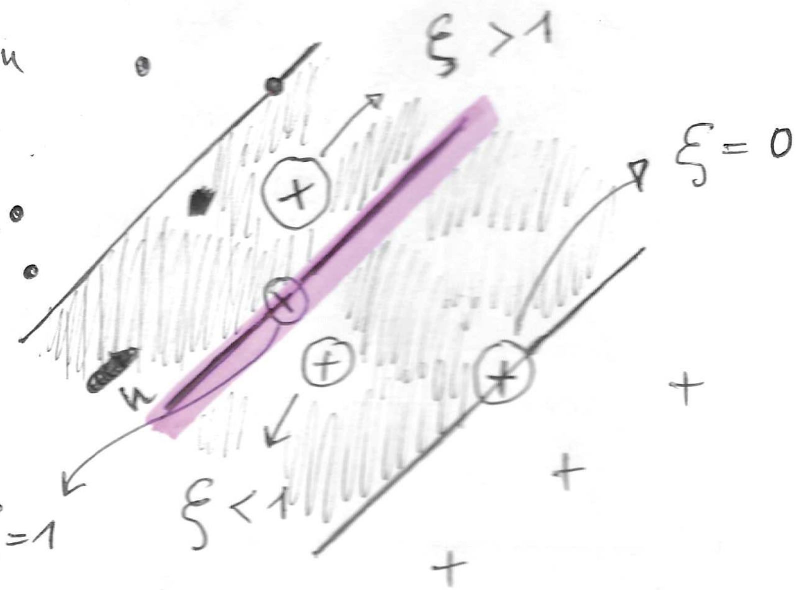
→ $\xi_n = 0$ if the point is in the right side

→ $0 < \xi_n \leq 1$ if point is inside the margin but in the correct side

→ $\xi_n > 1$ if the point is on wrong side of the boundary.

"RISK" that you take when the sample moves away from the safe region

You you want to reduce this variables (to mini-mize). But...



I can accept that some few points move away from the region by paying a cost (expressed by the slack variables). WITH SOFT MARGIN COSTRAINT I CAN ACCEPT POINTS IN THE WRONG REGION!

$$\boxed{t_n y(\vec{x}_n) \geq 1 - \xi_n} \quad \forall n = 1 \ldots N$$

IS A RELAXATION.

The new optimization problem is given by the terms!

$$\min \frac{1}{2}\|\vec{w}\|^2 + C\sum_{n=1}^{N} \xi_n$$

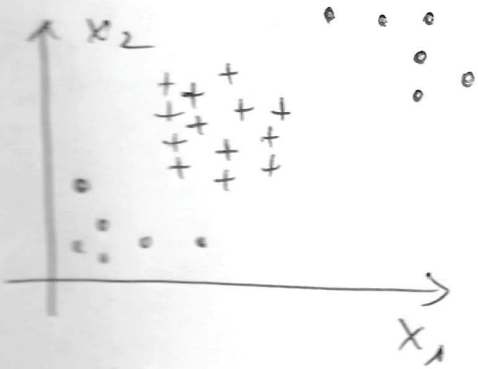
s.t. $t_n y(\vec{x}_n) \geq 1 - \xi_n \quad \forall n = 1 \dots N$

$$\xi_n \geq 0 \quad \forall n = 1 \dots N$$

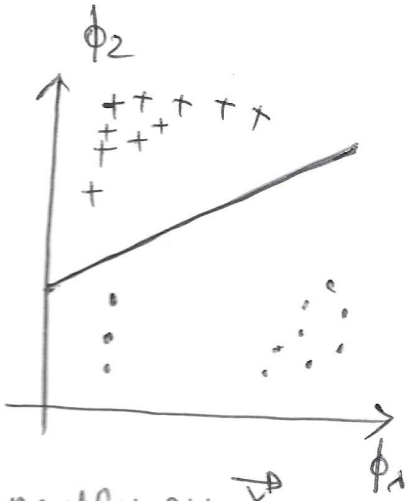
The solution is similar to the previous one, where most of a_n are 0; all these methods are robust to outliers, that are all points for which $a_n = 0$.

Basis function

Let's consider the following situation!



WE can do a transformation by using for instance, polar coordinates



So far we considered models working directly on $\vec{x}$. All the results hold if we consider a non linear transformation of the input space, $\phi(\vec{x})$ (BASIS FUNCTION).

Decision boundaries will be linear in the feature space ϕ and non-linear in the original space $\vec{x}$.

CLASSES THAT ARE LINEARLY SEPARABLE IN THE FEATURE SPACE ϕ MAY NOT BE SEPARABLE IN THE INPUT SPACE.

We can try different transformations (kernel) to see which is the best.