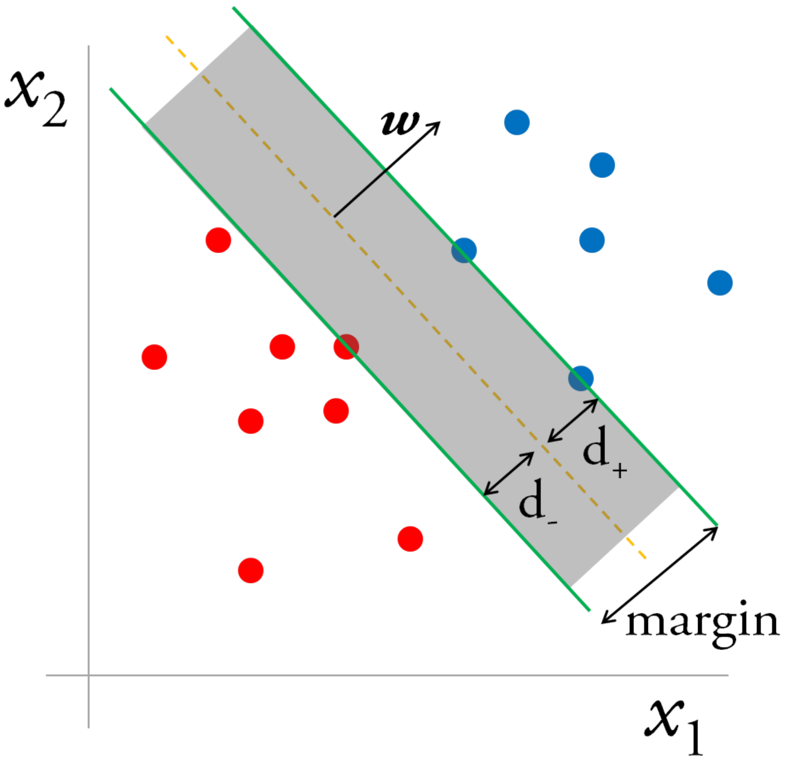
Support Vector Machines

The Support Vector Machines techniques aims to build a binary classifier by finding an hyperplane which is able to separate the data with the largest *margin* possible.



With SVMs we force our *margin* to be at least *something* in order to accept it, by doing that we restrict the number of possible dichotomies, and therefore if we're able to separate the points with a fat dichotomy (*margin*) then that fat dichotomy will have a smaller *VC* dimension then we'd have without any restriction. Let's do that.

Let be $\mathbf{x}_n$ the nearest data point to the *hyperplane* $\mathbf{w}^T\mathbf{x} = 0$ (just imagine a *line* in a 2-D space for simplicity), before finding the distance we just have to state two observations:

- There's a minor technicality about the *hyperplane* $\mathbf{w}^T\mathbf{x} = 0$ which is annoying , let's say I multiply the vector $\mathbf{w}$ by 1000000 , I get the *same* hyperplane! So any formula that takes $\mathbf{w}$ and produces the margin will have to have built-in *scale-invariance*, we do that by normalizing $\mathbf{w}$, requiring that for the nearest data point $\mathbf{x}_n$:

$$|\mathbf{w}^T\mathbf{x}_n| = 1 \tag{1}$$

  (So I just scale $\mathbf{w}$ up and down in order to fulfill the condition stated above, we just do it because it's *mathematically convenient*! By the way remember that 1 does *not* represent the Euclidean distance)

- When you solve for the margin, the w_1 to w_d will play a completely different role from the role of w_0 , so it is no longer convenient to have them on the same vector. We pull out w_0 from $\mathbf{w}$ and rename w_0 with b (for *bias*).

$$\begin{aligned} \mathbf{w} &= (w_1, \ldots, w_d) \\ w_0 &= b \end{aligned} \tag{2}$$

So now our notation is changed:

The *hyperplane* is represented by

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{3}$$

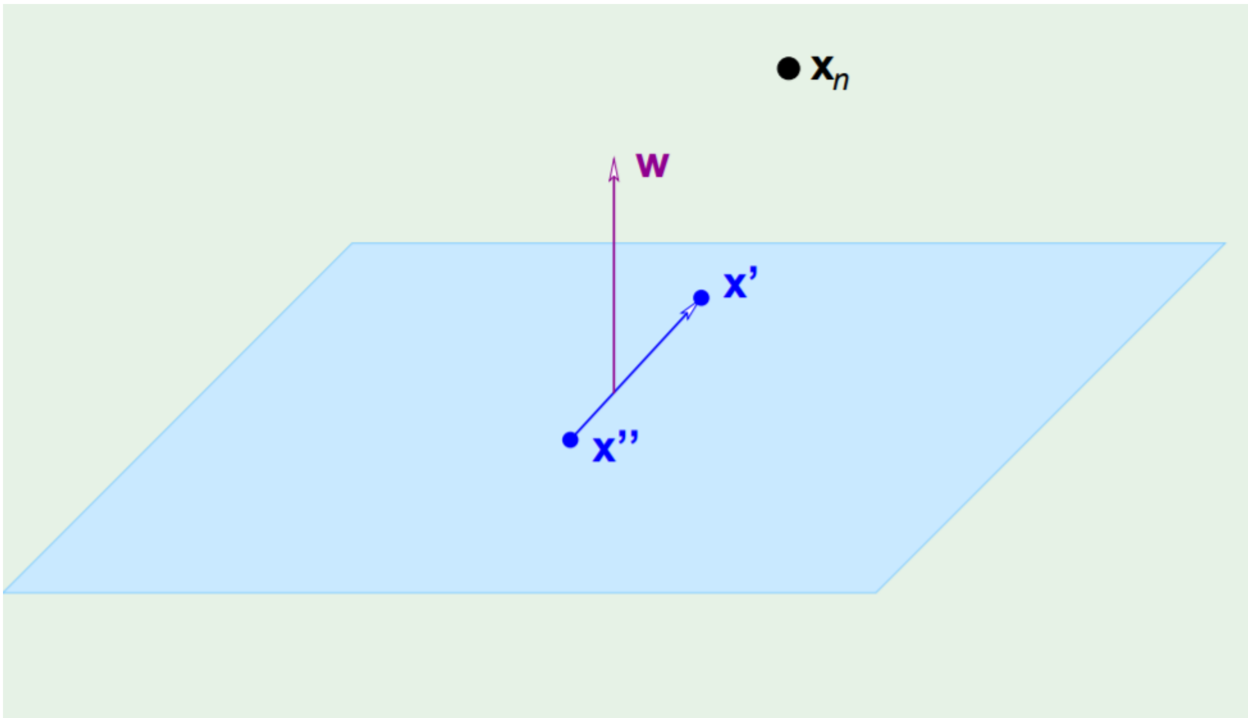
and our constraint becomes

$$|\mathbf{w}^T\mathbf{x}_n + b| = 1 \tag{4}$$

It's trivial to demonstrate that the vector $\mathbf{w}$ is orthogonal to the *hyperplane*, just suppose to have two point $\mathbf{x}'$ and $\mathbf{x}''$ belonging to the *hyperplane* , then $\mathbf{w}^T\mathbf{x}' + b = 0$ and $\mathbf{w}^T\mathbf{x}'' + b = 0$.

And of course $\mathbf{w}^T\mathbf{x}'' + b - (\mathbf{w}^T\mathbf{x}' + b) = \mathbf{w}^T(\mathbf{x}'' - \mathbf{x}') = 0$

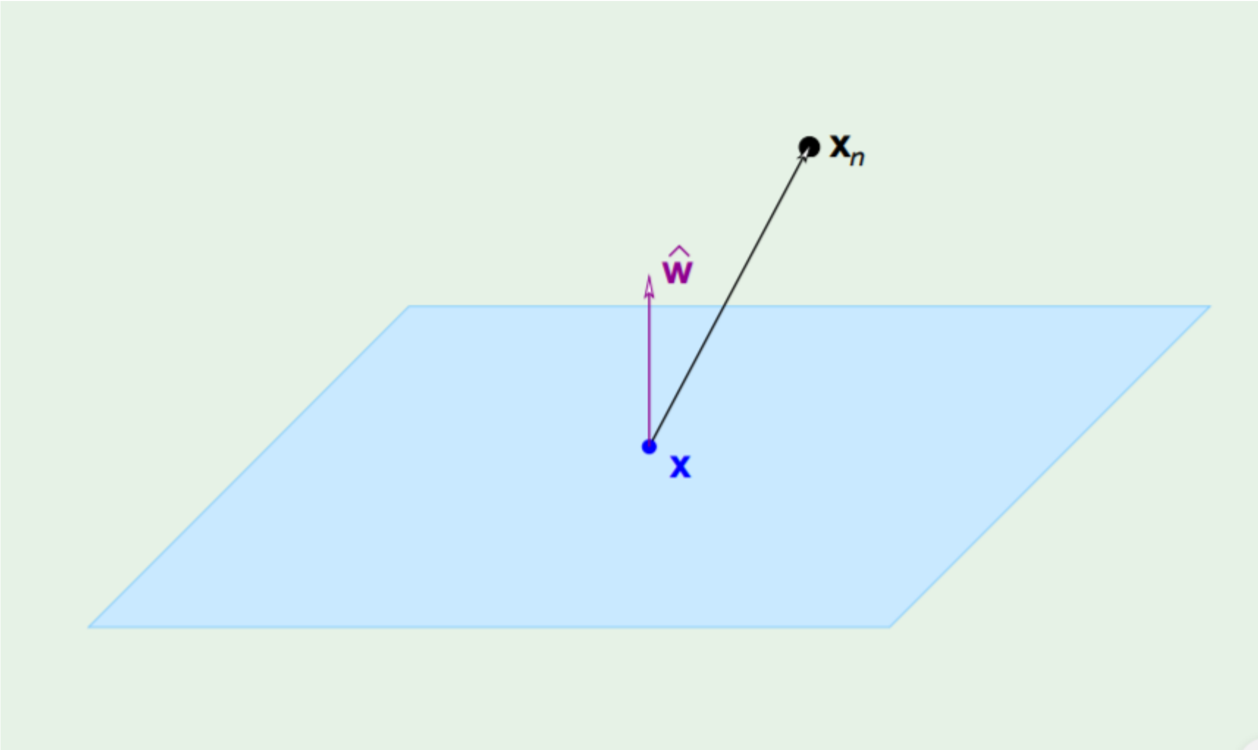
Since $\mathbf{x}'' - \mathbf{x}'$ is a vector which lays on the *hyperplane* , we deduce that $\mathbf{w}$ is orthogonal to the *hyperplane*.



Then the distance from $\mathbf{x}_n$ to the *hyperplane* can be expressed as a dot product between $\mathbf{x}_n - \mathbf{x}$ (where $\mathbf{x}$ is any point belonging to the plane) and the unit vector $\hat{\mathbf{w}}$, where $\hat{\mathbf{w}} = \frac{\mathbf{w}}{||\mathbf{w}||}$ (the distance is just the projection of $\mathbf{x}_n - \mathbf{x}$ in the direction of $\hat{\mathbf{w}}$!)

$$distance = |\ \hat{\mathbf{w}}^T(\mathbf{x}_n - \mathbf{x})\ | \tag{5}$$

(We take the absolute value since we don't know if $\mathbf{w}$ is facing $\mathbf{x}_n$ or is facing the other direction)



We'll now try to simplify our notion of *distance*.

$$distance = |\ \hat{\mathbf{w}}^T(\mathbf{x}_n - \mathbf{x})\ | = \frac{1}{||\mathbf{w}||}|\ \mathbf{w}^T\mathbf{x}_n - \mathbf{w}^T\mathbf{x}\ | \tag{6}$$

This can be simplified if we add and subtract the missing term b.

$$distance = \frac{1}{||\mathbf{w}||}|\ \mathbf{w}^T\mathbf{x}_n + b - \mathbf{w}^T\mathbf{x} - b\ | = \frac{1}{||\mathbf{w}||}|\ \mathbf{w}^T\mathbf{x}_n + b - (\mathbf{w}^T\mathbf{x} + b)\ | \tag{7}$$

Well, $\mathbf{w}^T\mathbf{x} + b$ is just the value of the equation of the plane...for a point *on* the plane. So without any doubt $\mathbf{w}^T\mathbf{x} + b = 0$, our notion of *distance* becomes

$$distance = \frac{1}{||\mathbf{w}||}|\ \mathbf{w}^T\mathbf{x}_n + b\ | \tag{8}$$

But wait...what is $|\ \mathbf{w}^T\mathbf{x}_n + b\ |$? It is the constraint that we defined at the beginning of our derivation!

$$|\mathbf{w}^T\mathbf{x}_n + b| = 1 \tag{9}$$

So we end up with the formula for the distance being just

$$distance = \frac{1}{||\mathbf{w}||} \tag{10}$$

Let's now formulate the optimization problem:

$$\begin{aligned} &\underset{w}{\operatorname{argmax}} \frac{1}{||\mathbf{w}||} \\ &\text{subject to} \min_{n=1,2,...,N} |\mathbf{w}^T\mathbf{x}_n + b| = 1 \end{aligned} \tag{11}$$

Since this is not a *friendly* optimization problem (the constraint is characterized by a minimum and an absolute, which are annoying) we are going to find an equivalent problem which is easier to solve. Our optimization problem can be rewritten as

$$\begin{aligned}&\underset{w}{\operatorname{argmin}}\ \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ &\text{subject to} \quad y_n \cdot (\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \text{for } n = 1, 2, \ldots, N\end{aligned} \tag{12}$$

where y_n is a variable that we introduce that will be equal to either $+1$ or -1 accordingly to its real target value (remember that this is a *supervised learning* technique and we know the real target value of each sample) . One could argue that the new constraint is actually different from the former one, since maybe the $\mathbf{w}$ that we'll find will allow the constraint to be *strictly* greater than 1 for every possible point in our dataset [$y_n(\mathbf{w}^T\mathbf{x}_n + b) > 1 \ \ \forall n$] while we'd like it to be *exactly* equal to 1 for *at least* one value of n. But that's actually not true! Since we're trying to minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ our algorithm will try to scale down $\mathbf{w}$ until $\mathbf{w}^T\mathbf{x}_n + b$ will touch 1 for some specific point n of the dataset.

So how can we solve this? This is a constraint optimization problem with inequality constraints, we have to derive the *Lagrangian* and apply the *KKT* (Karush–Kuhn–Tucker) conditions.

Objective Function:

We have to minimize

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N}\alpha_n(y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) \tag{13}$$

w.r.t. to $\mathbf{w}$ and b and maximize it *w.r.t.* the *Lagrange Multipliers* α_n

We can easily get the two conditions for the unconstrained part:

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n = 0 \qquad & \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n \\ \frac{\partial \mathcal{L}}{\partial b} = -\sum_{n=1}^{N}\alpha_n y_n = 0 \qquad & \sum_{n=1}^{N}\alpha_n y_n = 0\end{aligned} \tag{14}$$

And list the other *KKT* conditions:

$$\begin{aligned}y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1 \geq 0 \qquad \forall n \\ \alpha_n \geq 0 \qquad \forall n \\ \alpha_n(y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) = 0 \qquad \forall n\end{aligned} \tag{15}$$

Alert : the last condition is called the KKT *dual complementary condition* and will be key for showing that the SVM has only a small number of "support vectors", and will also give us our convergence test when we'll talk about the *SMO* algorithm.

Now we can reformulate the *Lagrangian* by applying some substitutions

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N} \alpha_n (y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) \tag{16}$$

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m$$

(if you have doubts just go to minute 36.50 of [this]() lecture by professor Yaser Abu-Mostafa at *Caltech*)

We end up with the *dual* formulation of the problem

$$\underset{\alpha}{\operatorname{argmax}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m \tag{17}$$

$$s.t. \qquad \alpha_n \geq 0 \quad \forall n$$

$$\sum_{n=1}^{N} \alpha_n y_n = 0$$

We can notice that the old constraint $\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$ doesn't appear in the new formulation since it is *not* a constraint on α , it was a constraint on $\mathbf{w}$ which is not part of our formulation anymore.

How do we find the solution? we throw this objective (which btw happens to be a *convex* function) to a *quadratic programming* package.

Once the *quadratic programming* package gives us back the solution we find out that a whole bunch of α are just 0 ! All the α which are not 0 are the ones associated with the so-called *support vectors* ! (which are just samples from our dataset)
They are called *support* vectors because they are the vectors that determine the width of the *margin* , this can be noted by observing the last *KKT* condition
$\{\alpha_n (y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) = 0 \quad \forall n\}$,
in fact either a constraint is active, and hence the point is a support vector, or its multiplier is zero.

Now that we solved the problem we can get both $\mathbf{w}$ and b.

$$\mathbf{w} = \sum_{\mathbf{x}_n \in \text{ SV}} \alpha_n y_n \mathbf{x}_n \tag{18}$$

$$y_n(\mathbf{w}^T\mathbf{x}_{n\in \text{SV}} + b) = 1$$

where $\mathbf{x}_{n\in \text{SV}}$ is any *support vector*. (you'd find the *same* b for every support vector)

But the coolest thing about *SVMs* is that we can rewrite our *objective functions*.
From

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m \tag{19}$$

to

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{20}$$

We can use *kernels* !! (if you don't know what I'm talking about read the *kernel* related question present somewhere in this document)

Finally we end up with the following equation for classifying *new points*:

$$\hat{y}(\mathbf{x}) = sign\left(\sum_{n=1}^{N} \alpha_n y_n k(\mathbf{x}, \mathbf{x}_n) + b\right) \tag{21}$$

The method described so far is called *hard-margin SVM* since the margin has to be satisfied strictly, it can happen that the points are not *linearly separable* in *any* way, or we just want to handle *noisy data* to avoid overfitting, so now we're going to briefly define another version of it, which is called *soft-margin SVM* that allows for few errors and penalizes for them.

We introduce *slack variables* ξ_n , this way we allow to *violate* the margin constraint but we add a *penalty* expressed by the distance of the misclassified samples from the hyperplane (samples correctly classified have $\xi_n = 0$).

We now have to

$$\begin{gathered}\text{Minimize} \ \ ||\mathbf{w}||_2^2 + C\sum_n \xi_n \\ \text{s.t.} \\ y_n(\mathbf{w}^T x_n + b) \geq 1 - \xi_n \ , \quad \forall n \\ \xi_n \geq 0 \ , \quad \forall n \end{gathered} \tag{22}$$

C is a coefficient that allows to trade-off bias-variance and is chosen by *cross-validation*.

And obtain the *Dual Representation*

$$\begin{gathered}\text{Maximize} \quad \mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m k(\mathbf{x}_n \mathbf{x}_m) \\ \text{s.t.} \\ 0 \leq \alpha_n \leq C \quad \ \forall n \\ \sum_{n=1}^{N} \alpha_n y_n = 0 \end{gathered} \tag{23}$$

if $\alpha_n \leq 0$ the point x_n is just correctly classified.

if $0 < \alpha_n < C$ the points lies *on the margin*. They are indeed Support Vectors.

if $\alpha_n = C$ the point lies *inside the margin*, and it can be either *correctly classified* ($\xi_n \leq 1$) or *misclassified* ($\xi_n > 1$)

Fun fact: When C is large, larger slacks penalize the objective function of SVM's more than when C is small. As C approaches infinity, this means that having any slack variable set to non-zero would have infinite penalty. Consequently, as C approaches infinity, all slack variables are set to 0 and we end up with a hard-margin SVM classifier.