# Sparse kernel machines

## Maximum margin classifiers

Consider $K=2$ classification using a linear model: $y(\vec{x}) = \vec{w}^T \vec{\phi}(\vec{x}) + b$

Training dataset: $\vec{x}_1, \dots, \vec{x}_N$

$t_1, \dots, t_N$ $\quad t_n \in \{-1, 1\}$

Thus new data points are classified by the sign of $y(\vec{x})$.

Consider a linearly separable dataset:

$\exists$ one or more set $\{\vec{w}, b\}$ s.t.

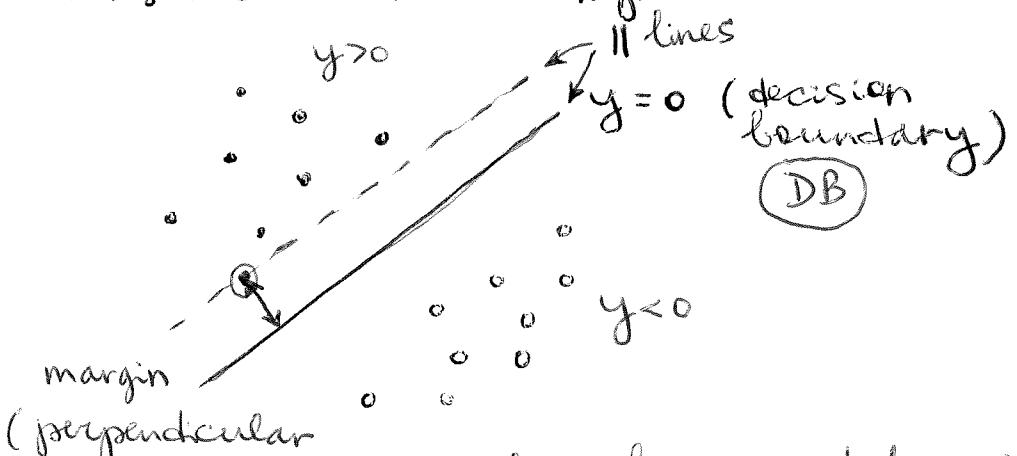
$$\begin{cases} y(\vec{x}_n) > 0 & \text{for all points with } t_n = 1 \\ y(\vec{x}_n) < 0 & \text{for all points with } t_n = -1 \end{cases}$$

$\Rightarrow$ always have $t_n y(\vec{x}_n) > 0$ $\quad n = 1, \dots, N$

There may be many $\{\vec{w}, b\}$, as was clear with e.g. the perceptron algorithm.

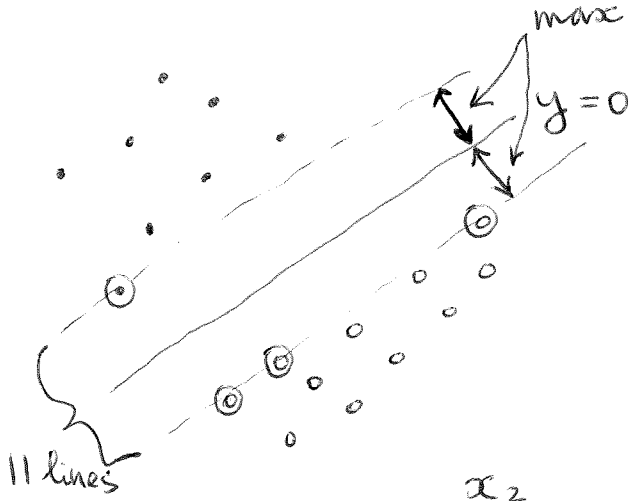
Which one is the best?

Consider an example:

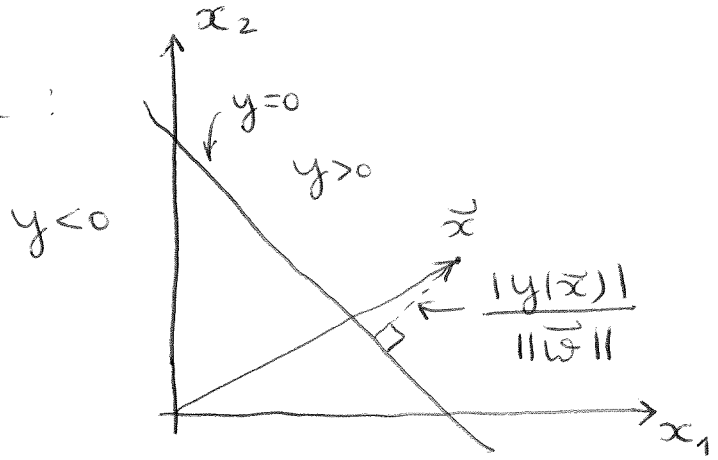



(perpendicular distance from the closest datapoint to $y=0$)

Idea: choose $\{\vec{w}, b\}$ which maximize the margin.




Recall:



Thus the distance from $\vec{x}_n$ to $y=0$ is given by:

$$\frac{t_n y(\vec{x}_n)}{\|\vec{w}\|} = \frac{t_n(\vec{w}^T\vec{\varphi}(\vec{x}_n)+b)}{\|\vec{w}\|}.$$

$t_n y(\vec{x}_n) > 0$, $\forall n$

The max margin solution is given by

$$\underset{\vec{w},b}{\operatorname{argmax}} \left\{ \frac{1}{\|\vec{w}\|} \underbrace{\min_n [t_n(\vec{w}^T\vec{\varphi}(\vec{x}_n)+b)]}_{\text{finds } \vec{x}_n \text{ closest to DB}} \right\} \quad (*)$$

Note that (*) is inv under $\begin{cases} \vec{w} \to k\vec{w} \\ b \to kb \end{cases}$ $\forall k>0$

Use this freedom to choose

$t_n(\vec{w}^T\vec{\varphi}(\vec{x}_n)+b) = 1$ for the closest $\vec{x}_n$.

Then $\underbrace{t_n(\vec{w}^T\vec{\varphi}(\vec{x}_n)+b) \geq 1}_{\text{canonical representation}}$, $\forall n$ (**)

$\vec{x}_n$'s for which the LHS = 1 are called active (there is at least one), while all other $\vec{x}_n$'s are called inactive. Then (*) becomes

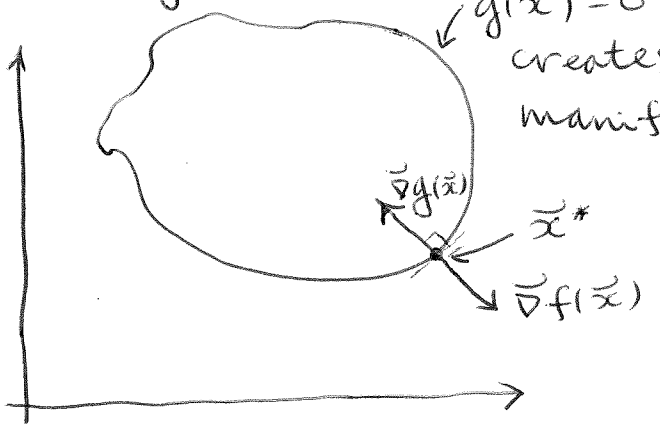
$$\underset{\vec{w},b}{\operatorname{argmax}} \left\{ \frac{1}{\|\vec{w}\|} \right\} \Rightarrow \underset{\vec{w},b}{\operatorname{argmin}} \left\{ \frac{1}{2}\|\vec{w}\|^2 \right\}$$

It's a quadratic programming problem: minimize a quadratic function subject to linear constraints (**)

This is best approached using Lagrange multipliers.

## Lagrange multipliers

Consider $\vec{x} = x_1, \ldots, x_D$ subject to a constraint $g(\vec{x}) = 0$




We want to maximize $f(\vec{x})$ subject to $g(\vec{x}) = 0$.

Consider $g(\vec{x} + \vec{\xi}) \simeq g(\vec{x}) + \vec{\xi}^T \cdot \vec{\nabla} g(\vec{x})$

($\vec{\xi}$: small; both points lie on the surface: $g(\vec{x}+\vec{\xi}) = g(\vec{x}) = 0$)

as $\|\vec{\xi}\| \to 0$, $\vec{\xi}^T \cdot \vec{\nabla} g(\vec{x}) = 0 \Rightarrow$ $\vec{\xi}$ is parallel to the surface in this limit, so $\vec{\nabla} g(\vec{x})$ is $\perp$ surface.

---

We seek $\vec{x}^*$ on the surface s.t. $f(\vec{x})$ is maximized. At $\vec{x}^*$, $\vec{\nabla} f(\vec{x})$ $\perp$ surface, otherwise we could move along the gradient & maximize $f(\vec{x})$ further.

Thus $\vec{\nabla} g \uparrow\downarrow \vec{\nabla} f$ or $\vec{\nabla} g \uparrow\uparrow \vec{\nabla} f$. In both cases,

$$\vec{\nabla} f + \lambda \vec{\nabla} g = 0 \quad \text{for some } \lambda \neq 0$$

(***)

↑ Lagrange multiplier

Introduce $L(\vec{x}, \lambda) = f(\vec{x}) + \lambda g(\vec{x})$, then

$\vec{\nabla} L = 0$ gives (***) and

$\frac{\partial L}{\partial \lambda} = 0 \Rightarrow g(\vec{x}) = 0$, original constraint

---

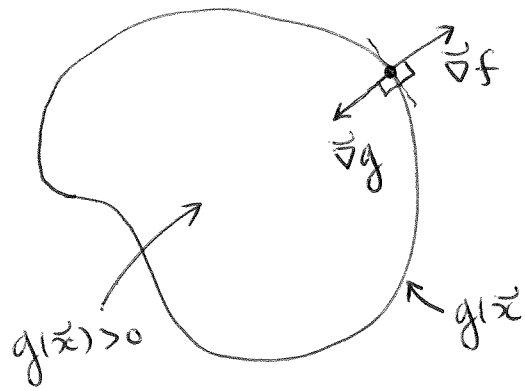
Now, consider $g(\vec{x}) \geq 0$ ← inequality constraint

If $g(\vec{x}) > 0 \Rightarrow \vec{\nabla} f(\vec{x}) = 0$, or

"constraint inactive" $\vec{\nabla} L = 0$ with $\lambda = 0$

If $g(\vec{x}) = 0 \Rightarrow \vec{\nabla} f + \lambda \vec{\nabla} g = 0$ as before, for some $\lambda \neq 0$.

"constraint active"




Note that $\vec{\nabla} f$ should point away from the $g(\vec{x}) > 0$ region since otherwise we could improve on it by stepping inside the region

So either $f(\vec{x})$ is maximized inside the region **or** on the boundary. In the latter case, $\vec{\nabla} f$ points away from the region.

Note that $\vec{\nabla} g$ points inward since $g(\vec{x}) > 0$ inside the boundary

This means that

$$\vec{\nabla} f = -\lambda \vec{\nabla} g \quad \text{for some } \underline{\lambda > 0}$$

In either case, $\lambda g(\vec{x}) = 0$.

Thus, maximizing $f(\vec{x})$ subject to $g(\vec{x}) \geq 0$ is the same as finding $\begin{cases} \frac{\partial L}{\partial \lambda} = 0, \\ \nabla L = 0 \end{cases}$, subject to

$$\begin{cases} g(\vec{x}) \geq 0, \\ \lambda \geq 0, \\ \lambda g(\vec{x}) = 0 \end{cases} \quad \text{KKT conditions}$$

If we want to <u>minimize</u> $f(\vec{x})$ subject to $g(\vec{x}) \geq 0$, we minimize

$$L(\vec{x}, \lambda) = f(\vec{x}) - \lambda g(\vec{x}) : \begin{cases} \vec{\nabla} L = 0 \\ \frac{\partial L}{\partial \lambda} = 0 \end{cases}$$

subject to $\lambda \geq 0$

To solve the constrained quadratic programming problem, we introduce

$$\mathcal{J}(\vec{w}, b, \vec{a}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{n=1}^{N} a_n \{t_n(\vec{w}^T \cdot \vec{\varphi}(\vec{x}_n) + b) - 1\},$$

where $\vec{a} = (a_1, \ldots, a_N)$ & $a_n \geq 0, \forall n$ are Lagrange multipliers.

$$\begin{cases} \frac{\partial \mathcal{J}}{\partial \vec{w}} = 0 \Rightarrow & \vec{w} = \sum_n a_n t_n \vec{\varphi}(\vec{x}_n), \\ \frac{\partial \mathcal{J}}{\partial b} = 0 \Rightarrow & \sum_n a_n t_n = 0. \end{cases}$$

Then $\mathcal{J} \to \tilde{\mathcal{J}}(\vec{a}) = \sum_n a_n + \frac{1}{2}\sum_{n,m} a_n t_n a_m t_m \times$

$$\times \vec{\varphi}^T(\vec{x}_n) \cdot \vec{\varphi}(\vec{x}_m) - \sum_{n,m} a_n t_n a_m t_m \vec{\varphi}^T(\vec{x}_m) \cdot \vec{\varphi}(\vec{x}_n) -$$

$$- b \sum_n a_n t_n = \sum_n a_n - \frac{1}{2}\sum_{n,m} a_n a_m t_n t_m \underbrace{\vec{\varphi}^T(\vec{x}_n) \cdot \vec{\varphi}(\vec{x}_m)}_{k(\vec{x}_n, \vec{x}_m)},$$

subject to $\begin{cases} a_n \geq 0, & \forall n \\ \sum_n a_n t_n = 0 \end{cases}$

We need to maximize $\tilde{\mathcal{J}}(\vec{a})$.

---

To classify new datapoints, need to evaluate the sign of

$$y(\vec{x}) = \vec{w}^T \cdot \vec{\varphi}(\vec{x}) + b = \sum_n a_n t_n \vec{\varphi}^T(\vec{x}_n) \cdot \vec{\varphi}(\vec{x}) + b =$$

$$= \sum_n a_n t_n k(\vec{x}_n, \vec{x}) + b \qquad (*)$$

The KKT conditions are:

$$\begin{cases} a_n \geq 0, \\ t_n y(\vec{x}_n) - 1 \geq 0, \\ a_n \{ t_n y(\vec{x}_n) - 1 \} = 0 \end{cases} \Rightarrow \quad \text{for each } n, \quad a_n = 0 \text{ OR } t_n y(\vec{x}_n) = 1 .$$

If $a_n = 0$, (*) is unaffected, so only the points for which $t_n y(\vec{x}_n) = 1$ matter. These are called <u>support vectors</u>, and lie on the max margin hyperplanes.

---

If we have $\vec{a}$ by solving the quadratic programming problem, we can find $b$:

$$t_n \Big( \sum_{m \in S} a_m t_m k(\vec{x}_n, \vec{x}_m) + b \Big) = 1 \quad , \forall n \in S$$

($S$: set of support vectors of size $N_S$)

Average over all $n \in S$ for numerical stability:

$$b = \frac{1}{N_S} \sum_{n \in S} \left[ t_n - \sum_{m \in S} a_m t_m k(\vec{x}_n, \vec{x}_m) \right]$$

Finally, $\vec{w} = \sum_{n \in S} a_n t_n \vec{\varphi}(\vec{x}_n)$.

---

The max margin classifier can be described by an error function (to be minimized)

$$\sum_{n=1}^{N} E_\infty (y(\vec{x}_n) t_n - 1) + \lambda \|\vec{w}\|^2 , \quad \text{where}$$

$$E_\infty(z) = \begin{cases} 0, & z \geq 0 \\ \infty, & z < 0 \end{cases}$$

$\lambda > 0$, otherwise its precise value is not important.

Note that max margin classifiers (SVMs) can draw non-linear boundaries in the $\vec{x}$-space, due to use of non-linear kernels (cf. Fig. 7.2 in Bishop).

## Overlapping class distributions

So far, we've assumed linear separability in some (potentially unknown explicitly) feature space $\vec{\varphi}(\vec{x})$. We need to modify the SVM to allow for training point misclassification in order to relax this assumption.

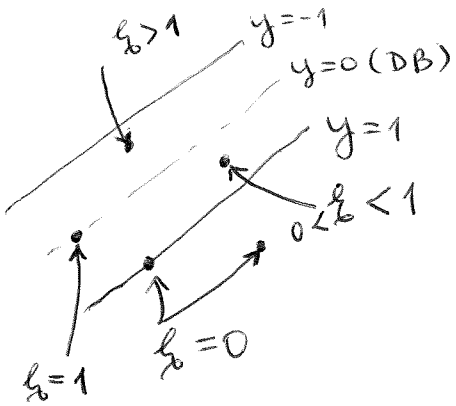
Introduce slack variables:

$$\xi_n > 0 \qquad n = 1, \ldots, N$$

s.t. $\xi_n = 0$ for all points on or inside the correct margin boundary &

$$\xi_n = \underbrace{|t_n - y(\vec{x}_n)|}_{\text{linear penalty}} \quad \text{for all points on the wrong side of the margin boundary}$$

Points on DB have $\xi_n = 1$, and all points with $\xi_n > 1$ are misclassified:




Now, $\underbrace{t_n\, y(\vec{x}_n) \geq 1, \ \forall n}_{\text{hard margin constraint}}$

is replaced with

$$t_n\, y(\vec{x}_n) \geq 1 - \xi_n, \ \forall n$$

$$\underbrace{\xi_n \geq 0}_{\text{soft margin constraint}}$$