SVM - Support Vector Machines

Pietro Gori

Associate Professor
Equipe IMAGES - Télécom Paris - IPParis
pietro.gori@telecom-paris.fr

Summary

Supervised Learning - Probabilistic framework

- X: input data x_i^j, random variable in $\mathcal{X} = \mathbb{R}^d$ with $i = 1, \ldots, n$ and $j = 1, \ldots, d$ where n and d are the number of observations and variables respectively
- Y: response (to predict), random variable in $\mathcal{Y} = \{C_1, \ldots, C_K\}$ (classification with K classes) or $\mathcal{Y} = \mathbb{R}$ (regression)
- p_{XY}: joint probability distribution of (X, Y), fixed but unknown
- $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$: i.i.d.samples drawn from p_{XY}
- $\mathcal{F}$: collection of classifiers $f \in \mathcal{F}$, where $f(\cdot) : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{L}$: loss function which measures the error of the classifier/model
  - Examples (classification) : $\mathcal{L}(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} 1, & \text{si } f(\mathbf{x}) \neq y, \\ 0, & \text{sinon.} \end{cases}$
  - Example (regression): $\mathcal{L}(\mathbf{x}, y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$

Goal: estimate from $\mathcal{D}_n$ the function $f \in \mathcal{F}$ which minimizes the risk (cost) function $R(f) = \mathbb{E}_P[\mathcal{L}(X, Y, f(X))]$

Estimate a classifier

We need to define:

- **input and output data space** ($\mathcal{X}$ and $\mathcal{Y}$)
- **type of classifier** ($\mathcal{F}$)
- **cost function** ($\mathcal{L}$) to minimize for finding the best f
- **minimization algorithm** for $\mathcal{L}$
- **method for model selection** to estimate hyper-parameters
- **method to evaluate performance**

Hyperplane

Definition

In $\mathbb{R}^d$ a **hyperplane** is an affine subspace of dimension $d-1$.

For instance in $\mathbb{R}^2$ a hyperplane is a line described by the equation:

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} = 0 \tag{1}$$

where $\beta_0, \beta_1, \beta_2$ are the parameters and all the $x_i = [x_{i1}, x_{i2}]$, for which Eq.1 holds, are points on the hyperplane.

We can use a hyperplane to separate the space into two halves:

$$\begin{aligned} \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} > 0 \\ \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} < 0 \end{aligned} \tag{2}$$

Separating Hyperplane - Binary classification

- In a binary classification problem, we can assign to the i-th observation $x_i \in \mathbb{R}^d$ a label of $y_i = 1$ if it belongs to the first class and $y_i = -1$ if it belongs to the second class.
- Then a separating hyperplane has the following property:

$$\begin{aligned} \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_d x_{id} > 0 \quad \text{if } y_i = 1 \\ \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_d x_{id} < 0 \quad \text{if } y_i = -1 \end{aligned} \tag{3}$$

- which can be written in the following way:

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_d x_{id}) > 0 \quad \forall i \tag{4}$$

- or equivalently using vectors $\beta = [\beta_0, ..., \beta_d]^T$ and $x_i = [x_{i1}, ..., x_{id}]$:

$$y_i(\beta_0 + \beta^T x_i) > 0 \quad \forall i \tag{5}$$

Separating Hyperplane - Binary classification

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_d x_{id}) > 0 \quad \forall i \tag{6}$$

- We can thus use the training set to estimate the hyperplane, namely all the coefficients $\{\beta j^*\}$, and then classify a new observation x_t in the validation (or test) set by simply looking at the sign of $\text{sign}(f(x_t)) = \beta_0^* + \beta_1^* x_{t1} + \beta_2^* x_{t2} + ... + \beta_d^* x_{td}$
- If $\text{sign}(f(x_t)) \geq 0$ then we classify x_t to the first class, otherwise, if $\text{sign}(f(x_t)) < 0$, to the second class
- We could also look to the *magnitude* of $f(x_t)$! If $f(x_t)$ is far from 0, it means that x_t is far from the hyperplane and thus we could be more confident about the classification

Separating Hyperplane - Binary classification

Problem: In general, if two groups can be well separated by a hyperplane, then there will exist an infinite number of similar hyperplanes ! (small translation or rotation...)

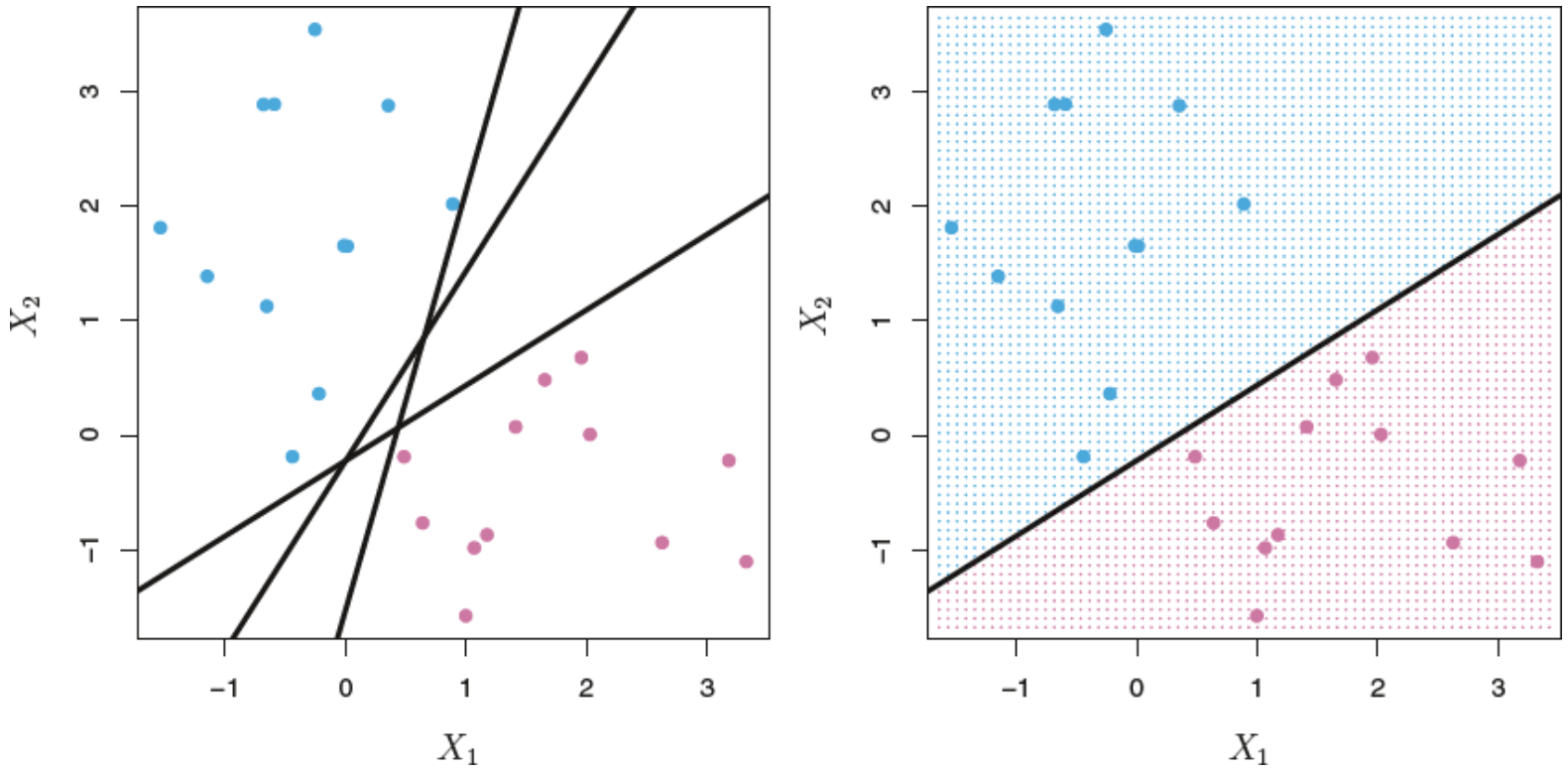


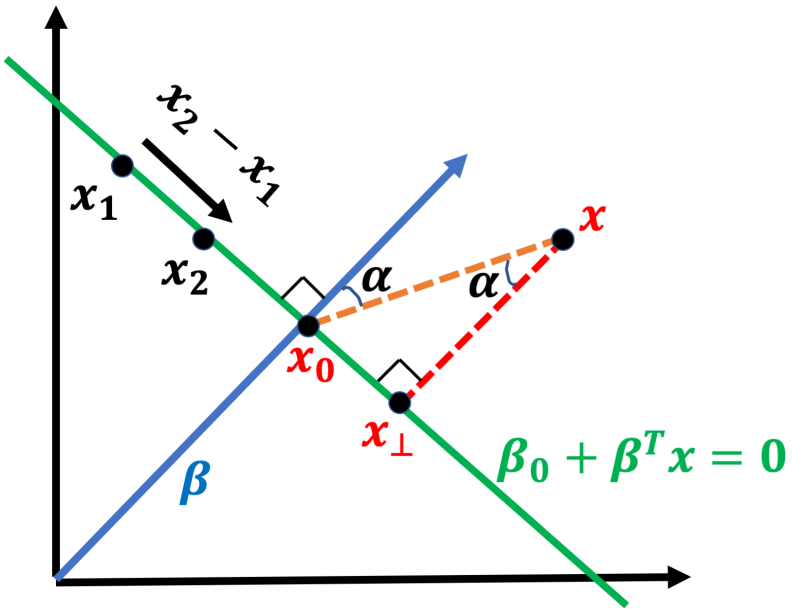
Figure 1: Left - Three possible separating hyperplanes in $\mathbb{R}^2$. Right - Separation made by the black hyperplane ($\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0$ and < 0) [3].

Recap vector algebra

Given a hyperplane defined by $f(x) = \beta_0 + \beta^T x = 0$, we have:

- For any two points x_1 and x_2 lying on the hyperplane, we have $f(x_1) = f(x_2) = 0$, which entails $\beta^T(x_1 - x_2) = 0$. Thus β is normal to the hyperplane
- $\forall x_0$ lying in the hyperplane, $f(x_0) = \beta_0 + \beta^T x_0 = 0 \rightarrow \beta^T x_0 = -\beta_0$
- The distance d of any point x from the hyperplane is $\frac{|f(x)|}{||\beta||_2}$. Proof:

$$d = ||x - x_\perp||_2 = ||x - x_0||_2 |\cos(\alpha)|$$
$$= |\frac{(x - x_\perp)^T}{||x - x_\perp||_2}(x - x_0)|$$



Since $(x - x_\perp)//\beta$, it results: $\frac{(x-x_\perp)^T}{||x-x_\perp||_2} = \pm\frac{\beta^T}{||\beta||_2}$

$$d = \frac{|\beta^T x - \beta^T x_0|}{||\beta||_2} = \frac{|\beta^T x + \beta_0|}{||\beta||_2} = \frac{|f(x)|}{||\beta||_2}$$

Summary

Perceptron (Rosenblatt)

- The perceptron algorithm finds a separating hyperplane by minimizing the distance of the misclassified points to the decision boundary
- This means that a misclassified point x_i with a $y_i = 1$ will have a $f(x_i) < 0$ and with a $y_i = -1$ will have a $f(x_i) > 0$.
- The distance of a misclassified point x_i to the decision boundary ($f(x) = 0$) is thus proportional to $-y_i f(x_i)$, which is always non-negative
- The idea is to associate a zero error with the x_i that are correctly classified and minimize the distances $-y_i f(x_i)$ for the misclassified x_i:

$$D(\beta, \beta_0) = -\sum_{i \in \mathcal{M}} y_i f(x_i) \tag{7}$$

- where $\mathcal{M}$ is the set of misclassified points

Perceptron (Rosenblatt)

$$D(\beta, \beta_0) = -\sum_{i \in \mathcal{M}} y_i f(x_i) = -\sum_{i \in \mathcal{M}} y_i (\beta_0 + \beta^T x_i) \tag{8}$$

- Assuming that $\mathcal{M}$ is fixed, we can compute the derivatives:

$$\begin{aligned} \frac{\partial D(\beta, \beta_0)}{\partial \beta} &= -\sum_{i \in \mathcal{M}} y_i x_i \\ \frac{\partial D(\beta, \beta_0)}{\partial \beta_0} &= -\sum_{i \in \mathcal{M}} y_i \end{aligned} \tag{9}$$

- The optimization technique usually used is *stochastic gradient descent*. The idea is to approximate the gradient of all summand (i.e. all i) with the gradient of a randomly chosen subset of $i \in \mathcal{M}$ (called "batch") or of a single i ("on-line") at each iteration

Perceptron (Rosenblatt)

Algorithm 1 Original "on-line" Perceptron pseudocode

Initialize $k = 1$, $\beta^1 = 0$, $\beta_0^1 = 0$, learning rate $\tau = 1$
Select max number of iterations K
for $j = 1$ to K **do**
  for $i = 1$ to N **do**
    if sign$(f(x_i)) \neq y_i$ **then**
      $\beta^{k+1} = \beta^k - \tau \nabla_\beta D_i = \beta^k + \tau y_i x_i$
      $\beta_0^{k+1} = \beta_0^k - \tau \nabla_{\beta_0} D_i = \beta_0^k + \tau y_i$
      $k = k + 1$
      if sign$(f(x_i)) = y_i \quad \forall i$ **then**
        break
      end if
    end if
  end for
end for

We can set $\tau = 1$ without loss in generality since if we multiply β and β_0 by $\frac{1}{\tau}$ the definition of the hyperplane does not change

Perceptron (Rosenblatt)

- We can notice that the contribution to the error $-y_i f(x_i)$ from a misclassified point is reduced after update of the coefficients β since:

$$-y_i(\beta^{k+1})^T x_i = -y_i(\beta^k)^T x_i - y_i^2 x_i^T x_i < -y_i(\beta^k)^T x_i \qquad (10)$$

- where we have used a dummy variable $x_0 = 1$ in order to include β_0 directly into β, set $\tau = 1$ and where $y_i^2 x_i^T x_i > 0$
- Of course this does not mean that: 1- the contribution to the error from the other misclassified points will reduce and 2- the change of the coefficients might have caused previously correctly classified x_i to become misclassified !
- However, the *perceptron convergence theorem* states that if the data are linearly separable then the algorithm converges to a separating hyperplane in a finite number of steps

Perceptron Convergence Proof

- By using $x_0 = 1$ in order to include β_0 directly into β, we can rewrite the Perceptron model as $\text{sign}(f(x_i)) = \text{sign}(\beta^T x_i) = \text{sign}(\beta \cdot x_i)$
- Let β^k be the parameter vector after the kth update which has occurred with the misclassified sample (x_i, y_i). This means that y_i and $f(x_i)$ had different signs and therefore that $y_i(\beta^k \cdot x_i) \leq 0$
- After updating the parameters we also have $\beta^{k+1} = \beta^k + y_i x_i$, with $\tau = 1$
- The Perceptron model will converge and correctly classify all samples in a finite number of steps if:

Hypotheses

1. Input data have finite size: $||x_i||_2 \leq R$, $\forall i$, $R \in \mathcal{R}^+$
2. Input data are linearly separable: $\exists u$ s.t. $||u||_2 = 1$ and $y_i(u \cdot x_i) \geq \gamma$, $\forall i$, $\gamma \in \mathcal{R}^+$.

Perceptron Convergence Proof

Hypotheses

1. Input data have finite size: $||x_i||_2 \leq R$, $\forall i$, $R \in \mathcal{R}^+$
2. Input data are linearly separable: $\exists u$ s.t. $||u||_2 = 1$ and $y_i(u \cdot x_i) \geq \gamma$, $\forall i$, $\gamma \in \mathcal{R}^+$.

- The first hypothesis means that all samples lie within a d-ball of radius R ($x_i \in \mathcal{R}^d$)
- The second hypothesis means that there exists a vector of parameters $\beta^* = u$ with which we can correctly classify all samples $\rightarrow y_i$ and $f(x_i)$ have all the same sign and thus their product is always positive
- Proof:

$$\beta^{k+1} \cdot u = \beta^k \cdot u + y_i(u \cdot x_i) \geq \beta^k \cdot u + \gamma \tag{11}$$

Perceptron Convergence Proof

$$\beta^{k+1} \cdot u = \beta^k \cdot u + y_i(u \cdot x_i) \geq \beta^k \cdot u + \gamma \tag{12}$$

- By induction, since $\beta^1 = 0$, we have that

$$\beta^2 \cdot u = \beta^1 \cdot u + y_i(u \cdot x_i) \geq \underbrace{\beta^1 \cdot u}_{=0} + \gamma \tag{13}$$

- Thus

$$\beta^3 \cdot u = \beta^2 \cdot u + y_i(u \cdot x_i) \geq \underbrace{\beta^2 \cdot u}_{\geq \gamma} + \gamma \geq 2\gamma \tag{14}$$

- and consequently, by induction, after k updates we have:

$$\beta^{k+1} \cdot u = \beta^k \cdot u + y_i(u \cdot x_i) \geq \beta^k \cdot u + \gamma \geq k\gamma \tag{15}$$

Perceptron Convergence Proof

- Then, by using the Cauchy-Schwartz inequality $|\langle a, b\rangle_2| \leq ||a||_2||b||_2$, we obtain:

$$||\beta^{k+1}||_2 \underbrace{||u||_2}_{=1} \geq \beta^{k+1} \cdot u \geq k\gamma \rightarrow ||\beta^{k+1}||_2^2 \geq k^2\gamma^2 \tag{16}$$

- we ignored the absolute value for $\beta^{k+1} \cdot u$ since both the number of updates k and γ are positive numbers and thus $\beta^{k+1} \cdot u$ must also be positive.
- This equation gives us lower bound for $||\beta^{k+1}||_2^2$, let's see if we can find also an upper bound

Perceptron Convergence Proof

- From the Perceptron update rule $\beta^{k+1} = \beta^k + y_i x_i$, we derive

$$||\beta^{k+1}||_2^2 = ||\beta^k + y_i x_i||_2^2 = ||\beta^k||_2^2 + \underbrace{y_i^2}_{=1} ||x_i||_2^2 + 2\underbrace{y_i(\beta^k \cdot x_i)}_{\leq 0} \tag{17}$$

$$\leq ||\beta^k||_2^2 + ||x_i||_2^2 \leq ||\beta^k||_2^2 + R^2 \tag{18}$$

- As before, by induction, we can notice that:

$$||\beta^2||_2^2 \leq \underbrace{||\beta^1||_2^2}_{=0} + R^2 \tag{19}$$

$$||\beta^3||_2^2 \leq \underbrace{||\beta^2||_2^2}_{\leq R^2} + R^2 \leq 2R^2 \tag{20}$$

$$||\beta^{k+1}||_2^2 \leq ||\beta^k||_2^2 + R^2 \leq kR^2 \tag{21}$$

Perceptron Convergence Proof

- We have thus found an upper bound for $||\beta^{k+1}||_2^2$, putting together with the previous lower bound, we obtain:

$$k^2\gamma^2 \leq ||\beta^{k+1}||_2^2 \leq kR^2 \rightarrow k \leq \frac{R^2}{\gamma^2} \tag{22}$$

- This shows that the number of updates k (i.e., misclassification errors) is bounded (if the two previous hypotheses are true !)
- What if the input data are not linearly separable ? $\rightarrow$ the algorithm will actually never converge !

Perceptron (Rosenblatt)

Limitations of the Perceptron algorithm

- If data are linearly separable there are many solutions. The perceptron finds one solution depending on the initial values of the coefficients and on the order of the data.
- The number of steps can be very long. The smaller the distance between the misclassified observations and the hyperplane, the longer the time to find it.
- If data are not linearly separable, the algorithm will not converge.
- It does not provide probabilistic outputs
- It is meant for binary classifications only

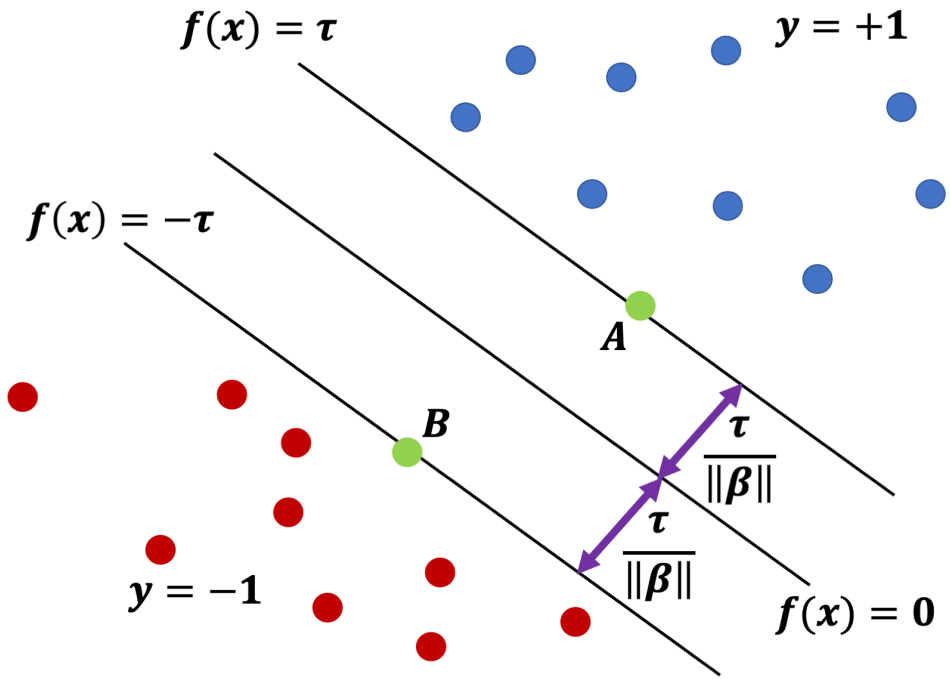
Possible improvement

- Enlarge the original space by using basis-function (i.e. non-linear transformations) of the original data $x_i \rightarrow \phi(x_i)$

Summary

Maximal margin hyperplane

- An elegant solution to the first problem (infinite number of solutions) is to add an additional constraint to the separating hyperplane
- The **maximal margin hyperplane** separates two (linearly separable) classes and **maximizes** the distance to the closest point (or points) from both classes, which are also equidistant from the hyperplane
- Let the hyperplanes $f(x) = \tau$ and $f(x) = -\tau$ be the ones passing through the closest points of the two classes (A and B respectively)



Maximal margin hyperplane

- We want to maximize the distance between the two hyperplanes, usually called the *margin* M, which are equidistant from $f(x) = 0$
- We can define the margin M as the distance between the point B and the hyper-plane $f(x) = \tau$ (or equivalently between A and $f(x) = -\tau$). Since the hyper-planes are parallel, and a commonly orthogonal vector is the unitary vector $\frac{\beta}{||\beta||_2}$ (see previous recap about vector algebra), we know that the point $B + M\frac{\beta}{||\beta||_2}$ will belong to the hyper-plane $f(x) = \tau$. Thus, we have $f(B + M\frac{\beta}{||\beta||_2}) = \tau$:

$$f(B + M\frac{\beta}{||\beta||_2}) = \beta^T(B + M\frac{\beta}{||\beta||_2}) + \beta_0 = \beta^T B + M\frac{\beta^T\beta}{||\beta||_2} + \beta_0$$

$$= \underbrace{\beta^T B + \beta_0}_{=-\tau} + M||\beta||_2 = \tau$$

$$M = \frac{2\tau}{||\beta||_2}$$

Maximal margin hyperplane

- The goal of the maximal margin hyperplane is therefore to maximize the margin M such that all data x_i are correctly classified $\rightarrow y_i f(x_i) \geq \tau \ \forall i$
- Mathematically this can be written as

$$\arg\max_{\beta,\beta_0} \frac{2\tau}{||\beta||_2} \text{ equivalent to } \arg\min_{\beta,\beta_0} \frac{1}{2\tau}||\beta||_2^2$$

- subject to $y_i(\beta^T x_i + \beta_0) \geq \tau \quad \forall i$
- Problem ! If (β_0^*, β^*) is a solution of the problem then also $(\alpha\beta_0^*, \alpha\beta^*)$ is a solution for any α since the maximal margin hyperplane would not change $\rightarrow (\beta^*)^T x_i + \beta_0^* = \alpha(\beta^*)^T x_i + \alpha\beta_0^* 0 = 0$

Maximal margin hyperplane

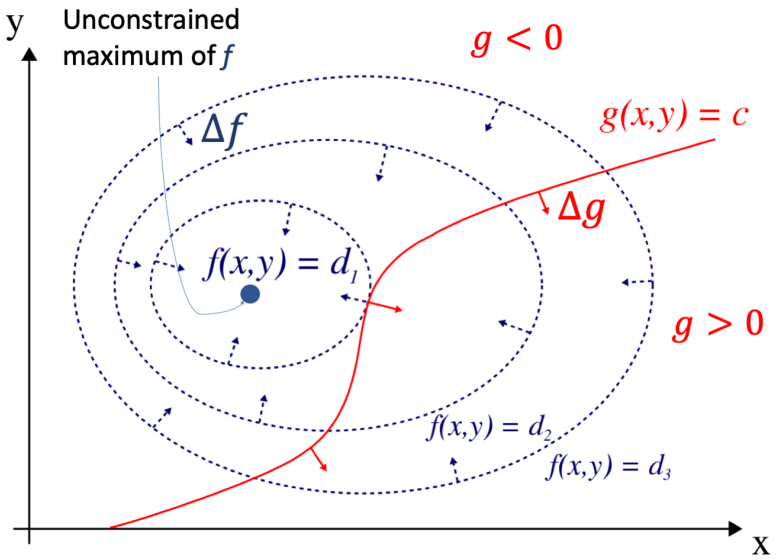
- To remove this ambiguity, a simple solution is to fix $\tau = 1$ which means that we set the scale of the problem or, equivalently, the scale of M
- The optimization criterion thus becomes:

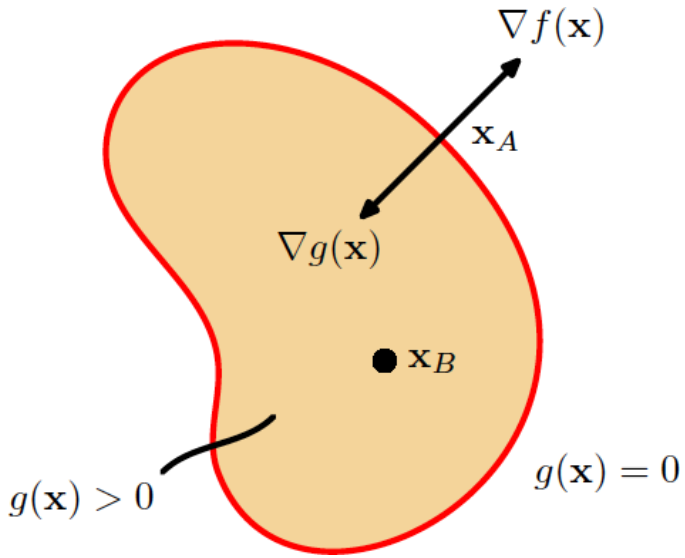
$$\arg\min_{\beta,\beta_0} \frac{1}{2}||\beta||_2^2 \quad \text{s.t. } y_i(\beta_0 + \beta^T x_i) \geq 1 \quad \forall i$$

- This is a constrained optimization problem with inequality constraint. Let's see how to solve it.

Lagrange Multipliers

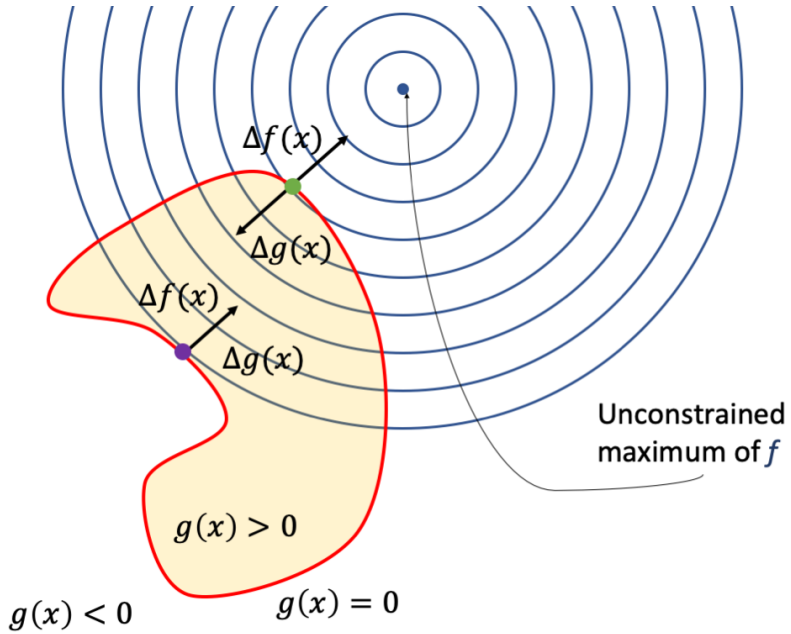
- When dealing with a constrained optimization problem with *inequality constraint* (i.e. $g(\mathbf{x}) \geq 0$ or $g(\mathbf{x}) \leq 0$), we have two possible cases:
  1. unconstrained maximum (or minimum) of $f(\mathbf{x})$ lies **within** the feasible region $g(\mathbf{x}) \geq 0$ (or $g(\mathbf{x}) \leq 0$) $\rightarrow$ $g(\mathbf{x})$ plays no role, stationary point occurs at $\nabla f(\mathbf{x}) = 0$ with $\lambda = 0$
  2. unconstrained maximum (or minimum) of $f(\mathbf{x})$ lies **outside** the feasible region $g(\mathbf{x}) \geq 0$ (or $g(\mathbf{x}) \leq 0$) $\rightarrow$ constrained stationary point are at the boundary $g(\mathbf{x}) = 0$. So $\nabla f // \nabla g$ but...





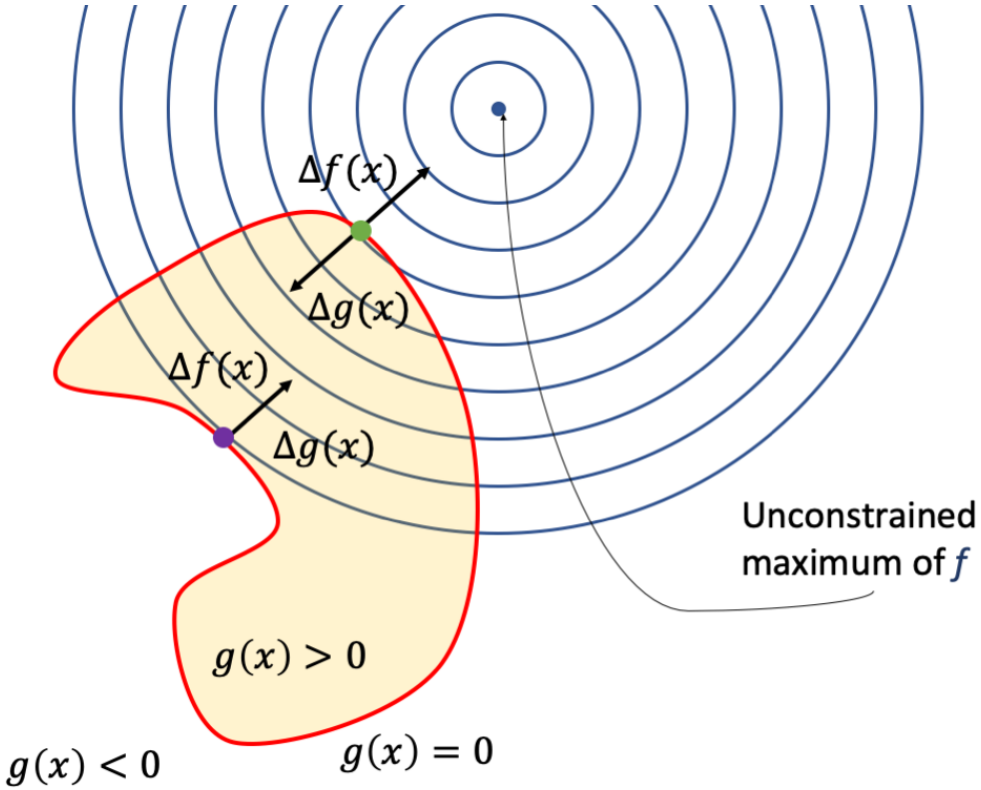
Lagrange Multipliers

- ... but we can directly find constrained local maxima (or minima).
- Two stationary points (green and purple) but the purple is **not** a constrained local maximum since $\nabla f(\mathbf{x})$ points towards the inside of the feasible region $\rightarrow$ there are certainly points with a greater $f(\mathbf{x})$
- The green point is a constrained local maximum since $\nabla f(\mathbf{x})$ points towards the outside of the feasible region $\rightarrow$ no points, within the feasible region, with a greater $f(\mathbf{x})$



Lagrange Multipliers

- The sign of the Lagrange multiplier is crucial !
- Constrained local maxima occur only when $\nabla f(\mathbf{x})$ and $\nabla g(\mathbf{x})$ have an opposite orientation, namely $\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x})$ for $\lambda > 0$



Lagrange Multipliers

To sum up:

- unconstrained maximum lies **within** the feasible region $\rightarrow$ stationary point of L with $\lambda = 0$ (i.e. $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$)
- unconstrained maximum lies **outside** the feasible region $\rightarrow$ constrained maximum lies on the boundary of the feasible region (i.e. $g(\mathbf{x}) = 0$) $\rightarrow$ stationary point of L with $\lambda > 0$ (this is why we put a plus in Eq.**??**))
- We can notice that in both cases $\lambda g(\mathbf{x}) = 0$ (either $\lambda = 0$ or $g(\mathbf{x}) = 0$). This gives us three conditions called Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned} \tag{23}$$

Lagrange Multipliers

- Given the optimization problem: $\max \quad f(\mathbf{x}) \quad s.t. \quad g(\mathbf{x}) \geq 0$
- Define the Lagrangian as: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
- We need to solve the following system of Eq.:

$$\begin{aligned} \nabla_{\mathbf{x}} L &= 0 \\ g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned} \tag{24}$$

- Plus negative definite constraint on $\nabla^2_{\mathbf{xx}} L$

Lagrange Multipliers

- Given the optimization problem: $\min \quad f(\mathbf{x}) \quad s.t. \quad g(\mathbf{x}) \geq 0$
- Define the Lagrangian as: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$
- We need to solve the following system of Eq.:

$$\begin{aligned} \nabla_{\mathbf{x}} L &= 0 \\ g(\mathbf{x}) &\geq 0 \\ \lambda &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned} \tag{25}$$

- Plus positive definite constraint on $\nabla^2_{\mathbf{xx}} L$

Maximal margin hyperplane

$$\begin{aligned} &\arg\min_{\beta,\beta_0} \frac{1}{2}||\beta||_2^2 \\ &\text{subject to } y_i(\beta_0+\beta^T x_i) \geq 1 \quad \forall i \end{aligned} \tag{26}$$

- This is a quadratic function subject to a set of linear inequality constraints. In order to solve it, we introduce Lagrangian multipliers $a_i \geq 0$ obtaining:

$$\mathcal{L}(\beta, \beta_0, a) = \frac{1}{2}||\beta||_2^2 - \sum_{i=1}^{N} a_i[y_i(\beta_0 + \beta^T x_i) - 1] \tag{27}$$

- where $a = \{a_i\}$. Note that the minus sign means that we are minimizing wrt β and β_0 but maximizing wrt a (inequality ≥ 0)

Maximal margin hyperplane

$$\mathcal{L}(\beta, \beta_0, a) = \frac{1}{2}||\beta||_2^2 - \sum_{i=1}^{N} a_i[y_i(\beta_0 + \beta^T x_i) - 1] \tag{28}$$

- Setting $\frac{\partial \mathcal{L}}{\partial \beta}$ and $\frac{\partial \mathcal{L}}{\partial \beta_0}$ equal to zero, we obtain:

$$\beta^* = \sum_{i=1}^{N} a_i y_i x_i \quad \text{and} \quad \sum_{i=1}^{N} a_i y_i = 0$$

- Substituting these in Eq.28, we obtain the *dual representation* (lower bound of Eq.26):

$$\arg\max_a \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j x_i^T x_j$$

$$\text{subject to } a_i \geq 0 \text{ and } \sum_{i=1}^{N} a_i y_i = 0$$

Maximal margin hyperplane

- The solution needs also to satisfy all the other constraints (KKT conditions):

$$\arg\max_{a} \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j x_i^T x_j$$

subject to

$$a_i \geq 0 \quad \sum_{i=1}^{N} a_i y_i = 0$$

$$\beta^* = \sum_{i=1}^{N} a_i y_i x_i \quad \sum_{i=1}^{N} a_i y_i = 0$$

$$a_i [y_i(\beta_0 + \beta^T x_i) - 1] = 0 \quad \forall i$$

Maximal margin hyperplane

From the previous Eq., we can notice that:

- Either $a_i = 0$ or $y_i f(x_i) = 1$. Only the points x_i which are closest to the decision boundary (ie on the margin $y_i f(x_i) = 1$) have $\alpha > 0$. They are called **support vectors**
- **Only support vectors are used to estimate β and β_0.** To estimate β_0^* we can notice that by substituting β^* into $f(x)$ we obtain $f(x) = \sum_{i=1}^{N} a_i y_i x^T x_i + \beta_0$
- If we consider only the support vectors ($y_i f(x_i) = 1$) we obtain:

$$y_j f(x_j) = y_j \left(\sum_{s \in \mathcal{S}} a_s y_s x_j^T x_s + \beta_0 \right) = 1 \tag{29}$$

- where $\mathcal{S}$ denotes the set of support vectors. If we then multiply everything by y_j we obtain:

$$\beta_0 = \frac{1}{N_{\mathcal{S}}} \sum_{j \in \mathcal{S}} \left(y_j - \sum_{s \in \mathcal{S}} a_s y_s x_j^T x_s \right) \tag{30}$$

Maximal margin hyperplane

From the previous Eq., we can notice that:

- Once estimated α_i^* and then β^* and β_0^*, we can build the optimal separating hyperplane as $f^*(x) = (\beta^*)^T x + \beta_0^*$ and classify any new observation x_t using $\text{sign} f^*(x)$
- Intuitively, the larger the margin in the training data, the better the separation in the validation/test data ! Remember that the hyperplane is at equal distance from the two classes.

Maximal margin hyperplane

The Maximal margin hyperplane is also called **linear SVM** (Support Vector Machine).

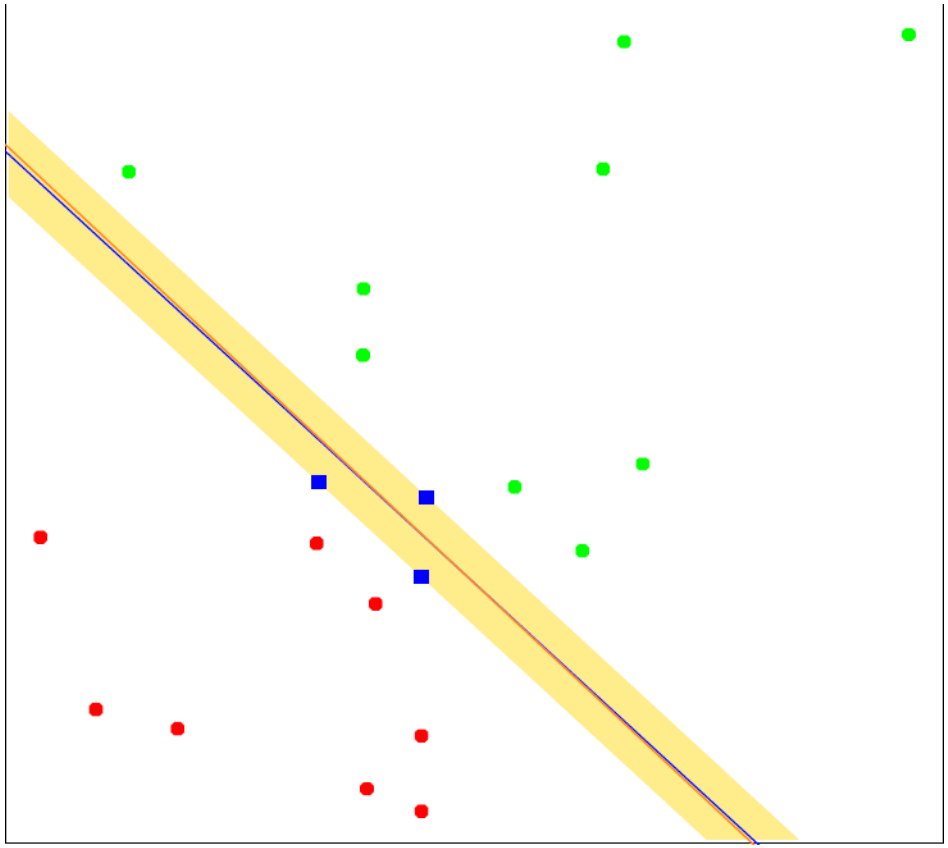


Figure 2: Example of maximal margin hyperplane. The blue dots are the support vectors which lie on the boundary of the margin. Image from [2].

Linear SVM VS LDA

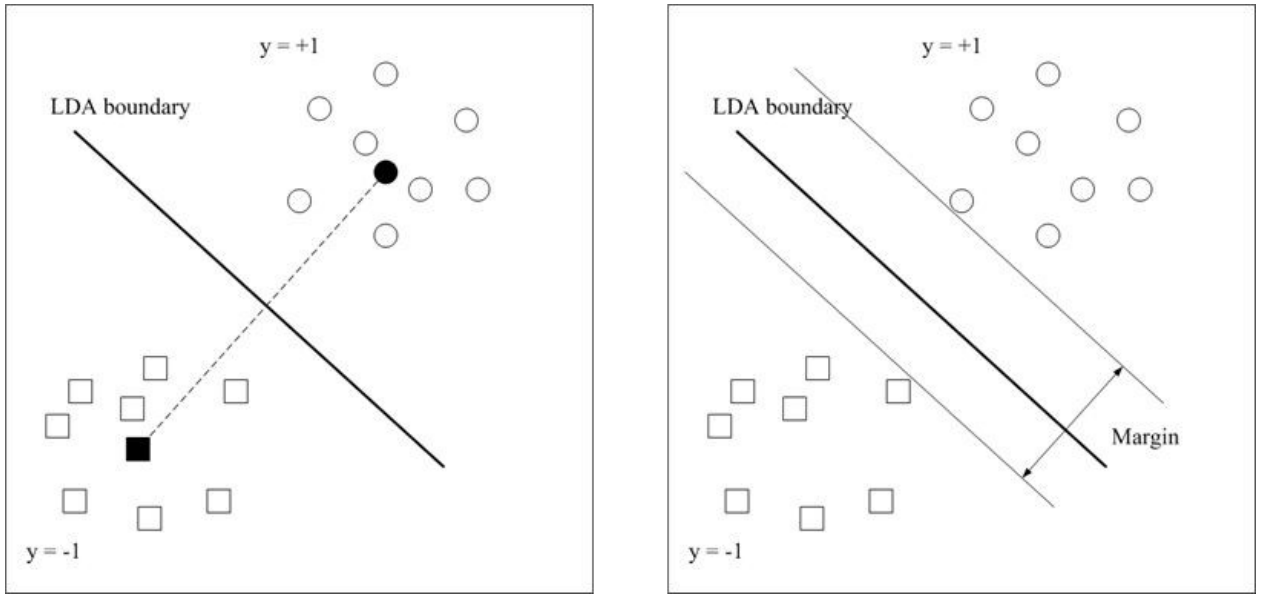
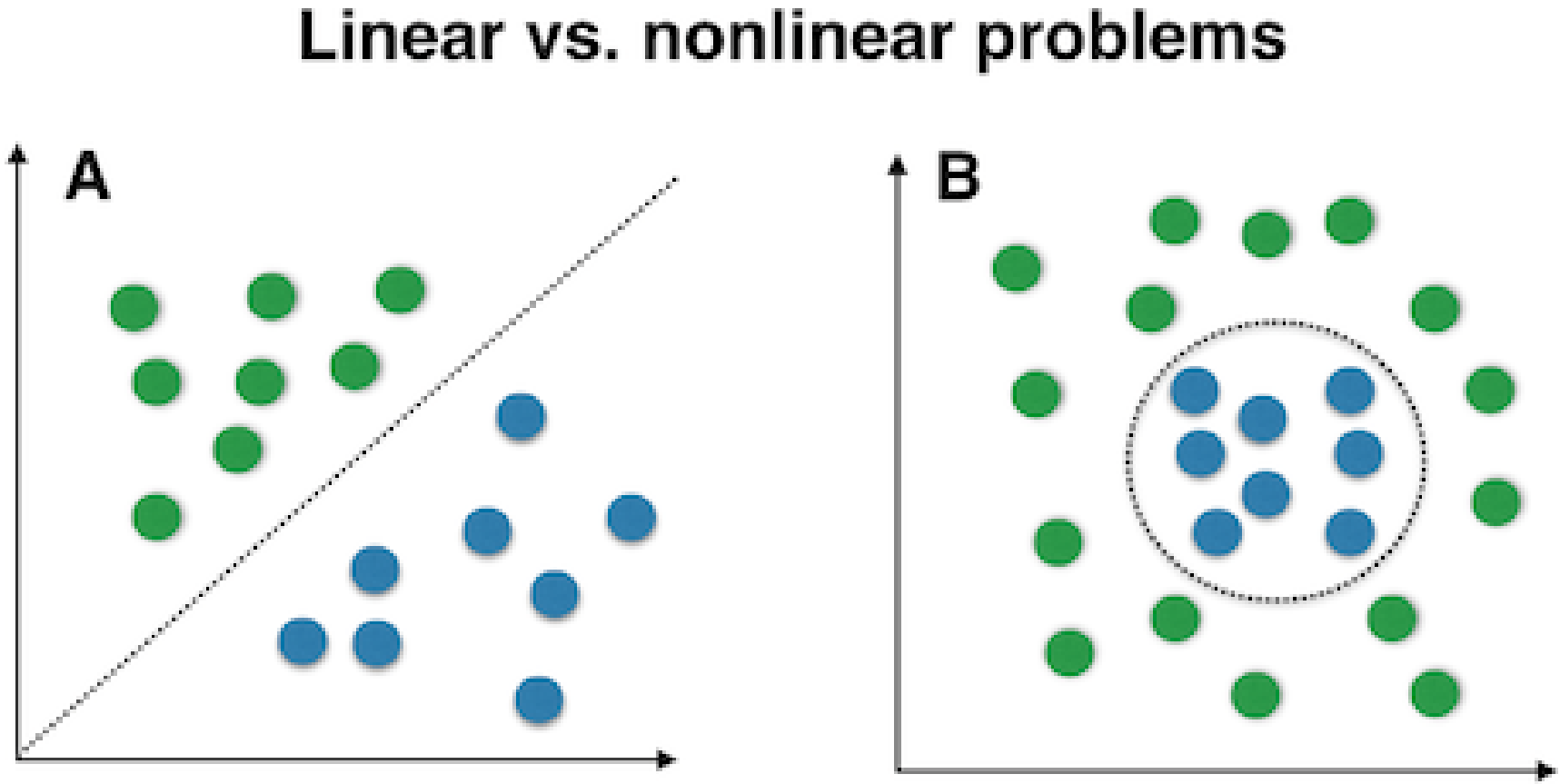


Figure 3: Comparison between LDA (left) and linear SVM (right).

- Decision boundary of linear SVM depends only on the support points, namely the points that count. $\rightarrow$ more robust to outliers
- Decision boundary of LDA depends on all data ! Even the ones far away from the decision boundary which might be outliers...
- If data follows Gaussian distributions, LDA is better. Otherwise, linear SVM is in general better.

Linear SVM

Does it always work linear SVM ?



Summary

Non-Linear SVM

- The idea is to map the data $x_i \rightarrow \phi(x_i)$ to a higher dimensional space where new data $\phi(x_i)$ might be linearly separable
- ϕ is a non-linear function in a high-dimensional space. This will probably make all computations more complicated... Do we really need to define and compute ϕ ?

$$\arg\max_a \sum_{i=1}^N a_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N a_i a_j y_i y_j \phi(x_i)^T \phi(x_j)$$

$$\text{subject to } a_i \geq 0 \quad \sum_{i=1}^N a_i y_i = 0$$

$$\beta^* = \sum_{i=1}^N a_i y_i \phi(x_i) \quad \sum_{i=1}^N a_i y_i = 0$$

$$a_i[y_i(\beta_0 + \phi(x_i)^T \beta) - 1] = 0 \quad \forall i$$

Non-Linear SVM

- Substituting $\beta^* = \sum_{i=1}^{N} a_i y_i \phi(x_i)$ into $a_i[y_i(\beta_0 + \phi(x_i)^T v\beta) - 1] = 0$, it results: $a_i[y_i(\beta_0 + \sum_{j=1}^{N} a_j y_j \phi(x_i)^T \phi(x_j)) - 1] = 0$
- Hence we only need to compute the inner products between couples of mapped points: $\langle \phi(x_i), \phi(x_j) \rangle = \phi(x_i)^T \phi(x_j)$
- **Kernel trick**: we can use a symmetric positive-definite function, called **kernel**, to define the inner product $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ (see Mercer's theorem)
- Examples of kernels are:
  - d-th degree polynomial: $k(x_i, x_j) = (1 + x_i^T x_j)^d$
  - radial basis (Gaussian): $k(x_i, x_j) = \exp(-\frac{||x_i - x_j||_2^2}{\sigma})$
  - sigmoid (neural network): $k(x_i, x_j) = \tanh(\gamma_1(x_i^T x_j) + \gamma_2)$

Non-Linear SVM

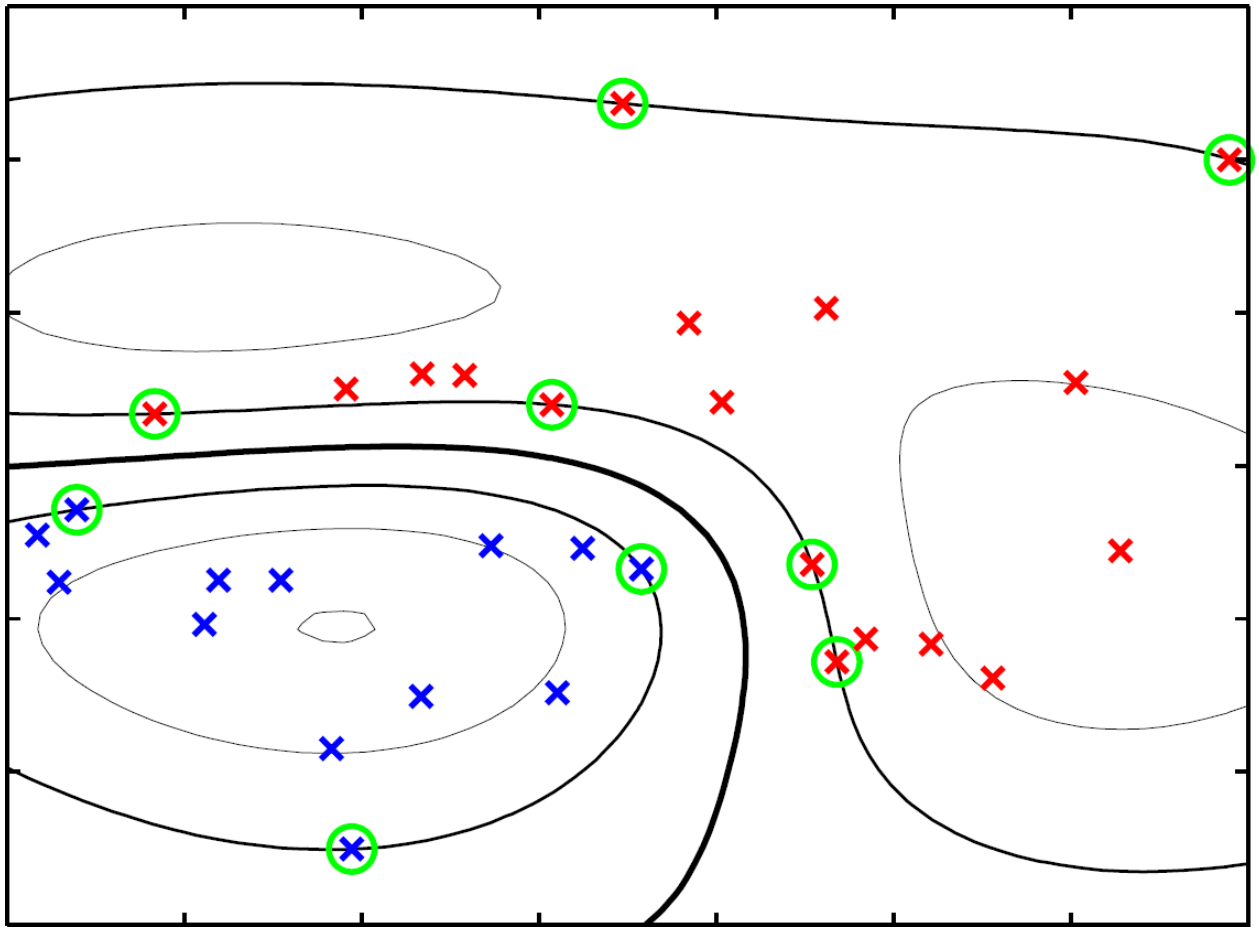


Figure 4: Example on non-linear SVM using a Gaussian kernel function. Grey lines show constant values of $f(x)$, the darkest line is the decision boundary and black lines describe the margin. Support vectors are highlighted in green [2].

Summary

SVM - Not-separable (overlapping) data

And what if the classes overlap (non-separable) ?

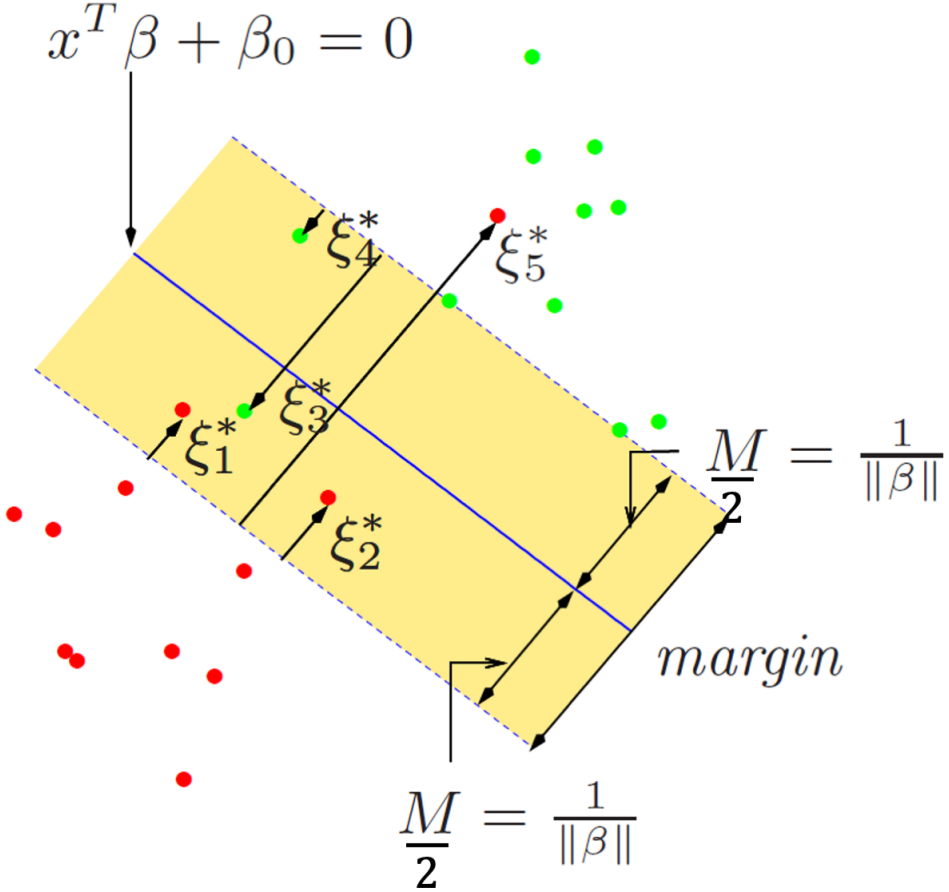
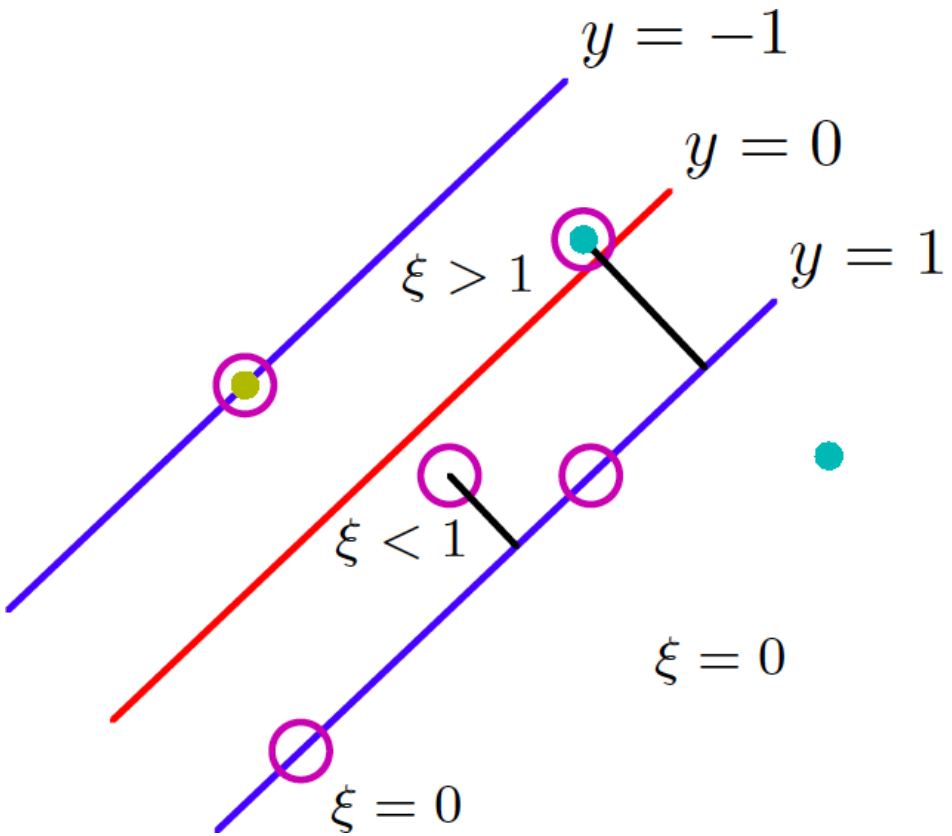


Figure 5: All points labeled with ξ^* are on the wrong side of their margin. Our goal is to maximize the margin M allowing at the same time some points to be on the wrong side of the margin.

SVM - Not-separable (overlapping) data

- This is called the **soft-margin** SVM where we introduce **slack variables** $\xi_i \geq 0 \quad \forall i$. They are defined using the *hinge loss*: $\xi_i = \max(0, 1 - y_i f(x_i))$ which results:



- $\xi_i = 0$ if x_i is correctly classified
- $\xi_i = 1 - y_i f(x_i)$ for other points. Hence, $0 < \xi_i \leq 1$ means margin violation and $\xi_i > 1$ means misclassification

SVM - Not-separable (overlapping) data

- Note that ξ_i is the smallest nonnegative number satisfying $y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i$ with $\xi_i \geq 0 \quad \forall i$
- To limit misclassifications and margin violations, we use an upper bound as regularization term $C \sum_{i=1}^{N} \xi_i$ where $C > 0$ is a trade-off parameter
- Note that the actual constraint would be: $y_i(\beta^T x_i + \beta_0) \geq \tau(1 - \xi_i)$, but as before we can set $\tau = 1$ for the points x_i that are closest to the decision boundary. Hence, the problem becomes as before:

$$\begin{aligned} &\arg\min_{\beta,\beta_0} \frac{1}{2}||\beta||_2^2 + C\sum_{i=1}^{N} \xi_i \\ &\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \\ &\xi_i \geq 0 \quad \forall i \end{aligned} \tag{31}$$

SVM - Not-separable (overlapping) data

- We can rewrite Eq. 31 using Lagrangian multipliers ($\alpha_i, \mu_i \geq 0$):

$$\mathcal{L}(\beta, \beta_0, \xi) = \frac{1}{2}||\beta||_2^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i[y_i(x_i^T\beta + \beta_0) - (1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i \tag{32}$$

- the parameter C controls the trade-off between the slack variables penalty and the margin. Small C means large (or soft) margin, large C means narrow (or hard) margin (since we focus more on driving the $\xi_i \to 0$). When $C \to \infty$, we obtain **optimal separating hyperplane**.
- Setting $\frac{\partial \mathcal{L}}{\partial \beta}$, $\frac{\partial \mathcal{L}}{\partial \beta_0}$ and $\frac{\partial \mathcal{L}}{\partial \xi_i}$ equal to zero, we obtain:

$$\beta^* = \sum_{i=1}^{N} a_i y_i x_i \ , \ \sum_{i=1}^{N} a_i y_i = 0 \ , \ \alpha_i = C - \mu_i \forall i \tag{33}$$

SVM - Not-separable (overlapping) data

- Even in this case, substituting these in Eq.32, we obtain the *dual representation* with the KKT conditions :

$$\arg\max_{a} \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j x_i^T x_j \tag{34}$$

$$\text{s.t. } 0 \leq a_i \leq C \ , \ \sum_{i=1}^{N} a_i y_i = 0 \tag{35}$$

$$a_i [y_i (x_i^T \beta + \beta_0) - (1 - \xi_i)] = 0 \tag{36}$$

$$y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \geq 0 \tag{37}$$

$$\mu_i \xi_i = 0 \quad \forall i \tag{38}$$

- Note that, even in this case, only the points x_i with a $\alpha_i > 0$ (support vectors) participate to the estimate of β^* and that we could also use the kernel trick to make it non-linear

SVM - Not-separable (overlapping) data

- Among the support vectors ($\alpha_i^* \neq 0$) some will lie on the edge of the margin ($\xi_i = 0$) and will be characterized by $0 \leq a_i^* < C$. The remainder ($\xi_i > 0$) will have $\alpha_i^* = C$ (see Eq.38 and Eq.33)
- To determine β_0 we can proceed as previously (see Eq.30)
- Once estimated α^* and then β^* and β_0^* we can classify any new point x using $G(x) = \text{sign}(f^*(x))$
- The hyper-parameters for SVM in the not-separable case are C and the parameter(s) of the kernel (when using it) $\rightarrow$ Cross-validation !

Relationships to other classification algorithms

- Using the regularized empirical risk minimization (ERM) framework, we can easily see the link between soft-margin SVM and other classification algorithms. In regularized ERM we minimize:

$$\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{data term}} + \lambda \underbrace{R(f)}_{\text{regularization term}} \tag{39}$$

- In the soft-margin SVM we use the hinge loss as L and the Tikhonov regularization (i.e., $l2$ norm on the model parameters) as R, obtaining:

$$\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i f(x_i)\right) + \lambda ||\beta||_2^2 \tag{40}$$

- The choice of the loss function L determines the classification method !

Relationships to other classification algorithms

- Ridge regression: $L(y_i, f(x_i)) = (y_i - f(x_i))^2$

$$\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda ||\beta||_2^2 \tag{41}$$

- Regularized logistic regression: $L(y_i, f(x_i)) = \ln(e^{-y_i f(x_i)}(1 + e^{f(x_i)})$

$$\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ln(e^{-y_i f(x_i)}(1 + e^{f(x_i)}) + \lambda ||\beta||_2^2 \tag{42}$$

- Regularized Perceptron: $L(y_i, f(x_i)) = \max(0, -y_i f(x_i))$

$$\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i f(x_i)) + \lambda ||\beta||_2^2 \tag{43}$$

Surrogate convex loss functions

Ideally, in a binary classification problem we would like to minimize the misclassification error - or 0-1 loss. However, it is non-convex and discontinuous, so gradient methods cannot be applied. This is why we optimize surrogate convex loss functions. More info in [4]

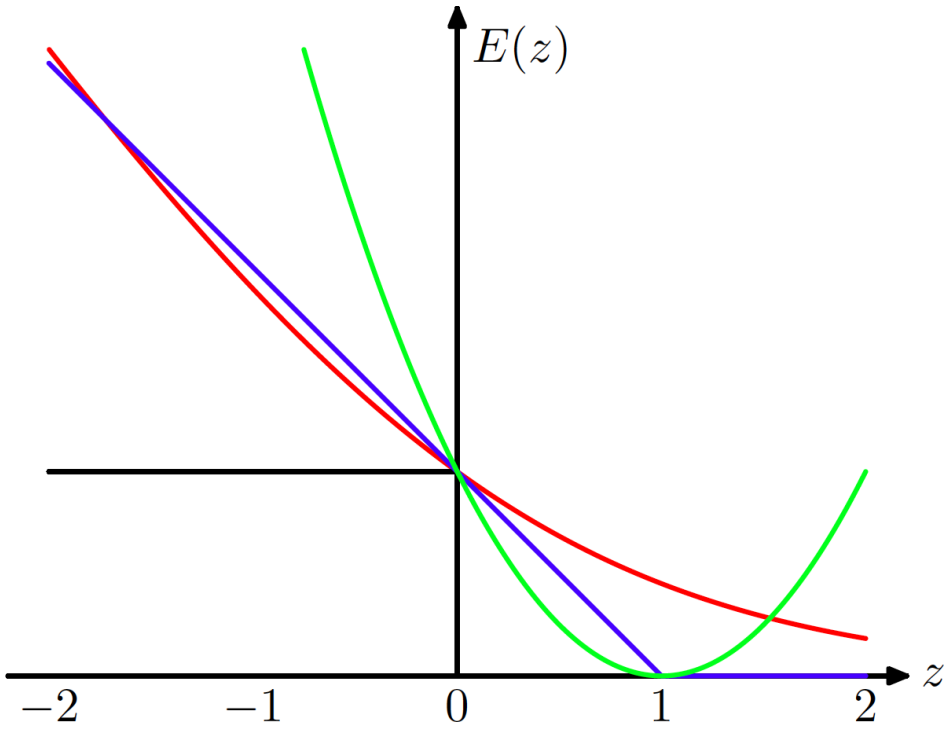


Figure 6: Blue: hinge function. Red: logistic function (rescaled by $\frac{1}{\ln(2)}$ so that it passes trough (0,1)). Black: misclassification error. Green: $L2$ error. From [1]

Surrogate convex loss functions

- The $L2$ error, or squared/quadratic error, gives too much importance on points that have been correctly classified but are far away from the decision boundary. Since we want to minimize the misclassification rate, a monotonically decreasing error function (such as hinge or logistic function) is a better choice
- The two best surrogate functions are the hinge loss (best convex approximation) and the logistic function (best smooth approximation)
- Please be aware that the logistic function is also sometimes called **binary cross entropy** (highly used in deep learning)
- Calling $y \in \{0, 1\}$ the ground truth label and $p(x; \beta) = P(Y = 1|X = x; \beta)$ the outcome of the logistic function, we define the discrete distribution $q \in \{p(x; \beta), 1 - p(x; \beta)\}$ and obtain:

$$H(y, q) = -\sum_s y_s \ln(q_s) = -y \ln(p(x; \beta)) - (1-y) \ln(1-p(x; \beta)) \quad (44)$$

- which is the cost function seen in the logistic regression lecture

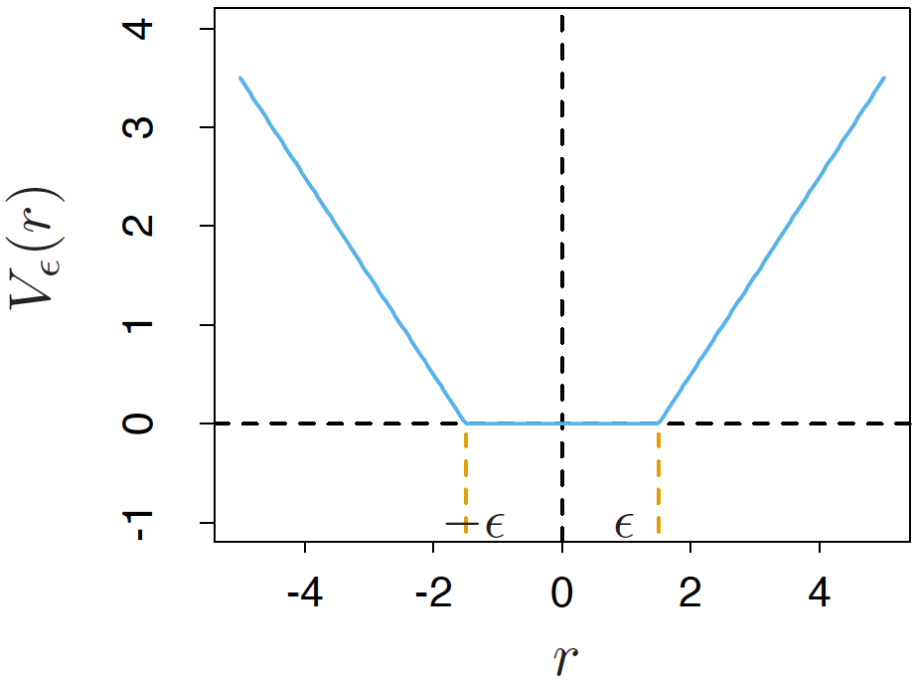
Summary

Multiclass SVM - $K > 2$ classes

- **One-versus-the-rest** approach: we train K different binary SVM where the k-th model considers class C_k versus all other classes. To make prediction for a sample x_t, we could then use the model that best classifies x_t, namely the one that places x_t the furthest away from the decision boundary
- *Problem*: different classifiers have been trained on different training sets (not same scale, number of samples, imbalanced data-sets).
- **One-versus-one** approach: we train $K(K-1)/2$ different binary classifiers on all possible pair of classes. To make prediction for a sample x_t, we classify it with all models and look for the class that has the highest score . Scores can be computed, for instance, as in soccer (3 for the winner, 0 for the loser).
- *Problem*: significant train and test time

Summary

SVR - Support Vector Regression Machine



- SVM can be adapted also for sparse **regression** where instead than using a $L2$ norm, one uses a ϵ-insensitive error function:

$$V_\epsilon(r) = \begin{cases} 0 & \text{if } |r| < \epsilon \\ |r| - \epsilon & \text{otherwise} \end{cases}$$

- where $r = y - f(x)$, y being the target

- Similarly to soft SVM, and using a linear regression model $f(x) = x^T\beta + \beta_0$, we can then write the cost function as:

$$\hat{f}(x) = \arg\min_{f \in \mathcal{F}} C \sum_{i=1}^{n} V_\epsilon(y_i - f(x_i)) + ||\beta||_2^2 \qquad (45)$$

SVR - Support Vector Regression Machine

- which can be rewritten as:

$$\beta^*, \beta_0^* = \arg\min_{\beta,\beta_0} C \sum_{i=1}^{n} \max\left(0, |y_i - (x_i^T \beta + \beta_0)| - \epsilon\right) + ||\beta||_2^2 \quad (46)$$

- where C is the (inverse) regularization hyper-parameter. Together with ϵ these are the two hyper-parameters to be set (usually set using Cross-Validation)
- The analogy between SVM and SVR is that in SVM points that are correctly classified and far away from the decision boundary are ignored. In SVR, the points that are ignored are the ones with small residuals ($|r| < \epsilon$)
- Of course, one can also use the kernel trick with SVR $x_i \to \phi(x_i)$

References

1. Bishop (2006). Pattern Recognition and Machine Learning. Springer.
2. Hastie, Tibshirani and Friedman (2009). The Element of Statistical Learning. Springer.
3. Tibshirani et al. (2013). An Introduction to Statistical Learning. Springer.
4. Shai Ben-David et al. (2012). Minimizing The Misclassification Error Rate Using a Surrogate Convex Loss. ICML