Machine Learning

Support Vector Machines

Fabio Vandin

November 24^{th}, 2023

Soft-SVM as RLM

Soft-SVM: solve

$$\min_{\mathbf{w},b,\xi} \left(\lambda ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right)$$

subject to $\forall i : y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$

Equivalent formulation with hinge loss:

$$\min_{\mathbf{w},b} \left(\lambda ||\mathbf{w}||^2 + L_S^{\text{hinge}}(\mathbf{w}, b) \right)$$

that is

$$\min_{\mathbf{w},b} \left(\lambda ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \ell^{\text{hinge}}((\mathbf{w}, b), (\mathbf{x}_i, y_i)) \right)$$

Note:

- $\lambda ||\mathbf{w}||^2$: ℓ_2 regularization
- $L_S^{\text{hinge}}(\mathbf{w}, b)$: empirical risk for hinge loss

Soft-SVM: Solution

We need to solve:

$$\min_{\mathbf{w},b}\left(\lambda||\mathbf{w}||^2 + \frac{1}{m}\sum_{i=1}^{m}\ell^{\text{hinge}}((\mathbf{w},b),(\mathbf{x}_i,y_i))\right)$$

where

$$\ell^{\text{hinge}}((\mathbf{w},b),(\mathbf{x},y)) = \max\{0, 1 - y(\langle \mathbf{w},\mathbf{x}\rangle + b)\}$$

How?

- standard solvers for optimization problems
- **Stochastic Gradient Descent**

SGD for Solving Soft-SVM

We want to solve

$$\min_{\mathbf{w}} \left(\frac{\lambda}{2} ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}_i \rangle\} \right)$$

Note: it's standard to add a $\frac{1}{2}$ in the regularization term to simplify some computations.

SGD algorithm:

$\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$;
for $t \leftarrow 1$ *to* T **do**
  $\eta^{(t)} \leftarrow \frac{1}{\lambda t}$; $\mathbf{w}^{(t)} \leftarrow \eta^{(t)} \boldsymbol{\theta}^{(t)}$;
  choose i uniformly at random from $\{1, \ldots, m\}$;
  if $y_i \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}_i \rangle < 1$ **then** $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + y_i \mathbf{x}_i$;
  else $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)}$;
return $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$;

Duality

We now present (Hard-)SVM in a different way which is very useful for *kernels*.
We want to solve

$$\mathbf{w}_0 = \min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 \text{ subject to } \forall i : y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$$

One can prove (details in the book!) that $\mathbf{w}$ that minimizes the function above is equivalent to find $\boldsymbol{\alpha}$ that solves the *dual problem*:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m : \boldsymbol{\alpha}\geq\mathbf{0}} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \right)$$

$m = |S|$

Duality

We now present (Hard-)SVM in a different way which is very useful for *kernels*.
We want to solve

$$\mathbf{w}_0 = \min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 \text{ subject to } \forall i : y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$$

One can prove (details in the book!) that $\mathbf{w}$ that minimizes the function above is equivalent to find $\boldsymbol{\alpha}$ that solves the *dual problem*:

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^m : \boldsymbol{\alpha} \geq \mathbf{0}} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \right)$$

Note:

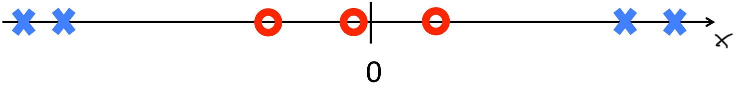
- solution is the vector $\boldsymbol{\alpha}$ which defines the support vectors $= \{\mathbf{x}_i : \alpha_i \neq 0\}$
- $\mathbf{w}_0$ can be derived from $\boldsymbol{\alpha}$ (see previous slides!)
- dual problem requires only to compute inner products $\langle \mathbf{x}_j, \mathbf{x}_i \rangle$, does not need to consider $\mathbf{x}_i$ by itself

SVM is a powerful algorithm, but still limited to linear models...
and linear models cannot always be used (directly)!

$\mathcal{X} = \mathbb{R}$

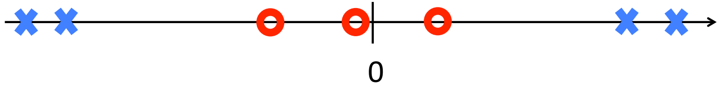
$\vec{x} = [x]$

Example



SVM is a powerful algorithm, but still limited to linear models...
and linear models cannot always be used (directly)!

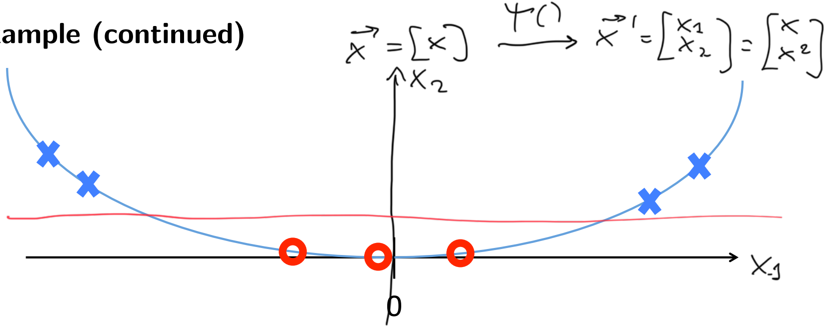
Example



We can:

- apply a nonlinear transformation $\psi()$ to each point in training set S first: $S' = ((\psi(\mathbf{x}_1), y_1), \ldots, (\psi(\mathbf{x}_m), y_m))$;
- learn a linear predictor $\hat{h}$ in the transformed space using S';
- make prediction for a new instance $\mathbf{x}$ as $\hat{h}(\psi(\mathbf{x}))$

Example (continued)



Kernel Trick for SVM

What if we want to apply a nonlinear transformation before using SVM?

Let $\psi()$ be the nonlinear transformation

i) Given a training set S: obtain training set S'

$$S = \{ (\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m) \}$$

$$\downarrow$$

$$S' = \{ (\psi(\vec{x}_1), y_1), \ldots, (\psi(\vec{x}_m), y_m) \}$$

ii) Learn a model with SVM using S'; let h_{svm} the model we learned

iii) Given $\vec{x} \in \mathcal{X}$, the prediction is $h_{svm}(\psi(\vec{x}))$

Kernel Trick for SVM

What if we want to apply a nonlinear transformation before using SVM?

Let $\psi()$ be the nonlinear transformation

Considering the dual formulation $\Rightarrow$ we only need to be able to compute $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ for some $\mathbf{x}$,$\mathbf{x}'$.

Definition

A *kernel function* is a function of the type:

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

where $\psi(\mathbf{x})$ is a transformation of $\mathbf{x}$.

Kernel Trick for SVM

What if we want to apply a nonlinear transformation before using SVM?

Let $\psi()$ be the nonlinear transformation

Considering the dual formulation $\Rightarrow$ we only need to be able to compute $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ for some $\mathbf{x}$,$\mathbf{x}'$.

Definition

A *kernel function* is a function of the type:

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

where $\psi(\mathbf{x})$ is a transformation of $\mathbf{x}$.

Intuition: we can think of K_ψ as specifying *similarity* between instances and of ψ as mapping the domain set $\mathcal{X}$ into a space where these similarities are realized as dot products.

Kernel Trick for SVM(2)

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

It seems that to compute $K_\psi(\mathbf{x}, \mathbf{x}')$ requires to be able to compute $\psi(\mathbf{x})$...

Not always... sometimes we can compute $K_\psi(\mathbf{x}, \mathbf{x}')$ without computing $\psi(\mathbf{x})$!

Kernel: Example

$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d$

Consider $\mathbf{x} \in \mathbb{R}^d$

$$\psi(\mathbf{x}) = (\overbrace{1, x_1, x_2, \ldots, x_d}^{d+1}, \underbrace{x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d}_{d^2})^T$$

What is the dimension of $\psi(\vec{x})$? $1+d+d^2$

Kernel: Example

Consider $\mathbf{x} \in \mathbb{R}^d$

$$\psi(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d)^T$$

The dimension of $\psi(\mathbf{x})$ is $1 + d + d^2$.

$$K_\psi(\vec{x}, \vec{x}') = \left\langle \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1x_1 \\ x_1x_2 \\ \vdots \\ x_dx_d \end{bmatrix}, \begin{bmatrix} 1 \\ x_1' \\ \vdots \\ x_d' \\ x_1'x_1' \\ x_1'x_2' \\ \vdots \\ x_d'x_d' \end{bmatrix} \right\rangle = 1 + (x_1x_1' + \ldots + x_dx_d') + x_1x_1x_1'x_1' + x_1x_2x_1'x_2' + \ldots + x_dx_dx_d'x_d'$$

$$= 1 + \underbrace{\sum_{i=1}^{d} x_ix_i'}_{\langle \vec{x}, \vec{x}' \rangle} + \sum_{i=1}^{d}\sum_{j=1}^{d} x_ix_jx_i'x_j'$$

$$\left(\sum_{i=1}^{d} x_ix_i'\right)\left(\sum_{j=1}^{d} x_jx_j'\right) = (\langle \vec{x}, \vec{x}' \rangle)^2$$

Kernel: Example

Consider $\mathbf{x} \in \mathbb{R}^d$

$$\psi(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d)^T$$

The dimension of $\psi(\mathbf{x})$ is $1 + d + d^2$.

$$\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j x_i' x_j'$$

Note that

$$\sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j x_i' x_j' = \left(\sum_{i=1}^{d} x_i x_i' \right) \left(\sum_{j=1}^{d} x_j x_j' \right) = \left(\langle \mathbf{x}, \mathbf{x}' \rangle \right)^2$$

therefore

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \langle \mathbf{x}, \mathbf{x}' \rangle + \left(\langle \mathbf{x}, \mathbf{x}' \rangle \right)^2$$

We have:

$$\psi(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d)^T$$

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \langle \mathbf{x}, \mathbf{x}' \rangle + (\langle \mathbf{x}, \mathbf{x}' \rangle)^2$$

Computation of $\psi(\vec{x})$: how many operations? $\mathcal{O}(d^2)$ (starting from $\vec{x}$)

Computation of $K_\psi(\vec{x})$: how many operations?

If I first compute $\psi(\vec{x})$ and $\psi(\vec{x}')$, and then compute the product $\langle \psi(\vec{x}), \psi(\vec{x}') \rangle$: $\mathcal{O}(d^2)$

If we use $K_\psi(\vec{x}, \vec{x}') = 1 + \langle \vec{x}, \vec{x}' \rangle + (\langle \vec{x}, \vec{x}' \rangle)^2$: $\mathcal{O}(d)$

We have:

$$\psi(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1x_1, x_1x_2, x_1x_3, \ldots, x_dx_d)^T$$

$$K_\psi(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle = 1 + \langle \mathbf{x}, \mathbf{x}' \rangle + \left(\langle \mathbf{x}, \mathbf{x}' \rangle \right)^2$$

Observation

Computing $\psi(\mathbf{x})$ requires $\Theta\left(d^2\right)$ time; computing $K_\psi(\mathbf{x}, \mathbf{x}')$ from the last formula requires $\Theta\left(d\right)$ time

When $K_\psi(\mathbf{x}, \mathbf{x}')$ is efficiently computable, we don't need to explicitly compute $\psi(\mathbf{x})$
$\Rightarrow$ *kernel trick*