Support Vector Machines (2/2)

Dr. Víctor Uc Cetina

Universität Hamburg

Content

1. Kernels
2. The Non-Separable Case
3. The SMO Algorithm
4. The LibSVM Tool

Kernels

- Back in our discussion of linear regression, we had a problem in which the input x was the living area of a house, and we considered performing regression using the features x, x^2 and x^3 to obtain a cubic function.
- Now, let ϕ denote a feature mapping, which maps from the attributes to the features. For instance, we can have

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

Kernels

- Rather than applying SVMs using the original input attributes x, we may instead want to learn using some features $\phi(x)$.
- We simply need to go over our previous algorithm, and replace x everywhere in it with $\phi(x)$.
- Since the algorithm can be written entirely in terms of the inner products $\langle x, z \rangle$, this means that we would replace all those inner products with $\langle \phi(x), \phi(z) \rangle$.

Kernels

- Specifically, given a feature mapping ϕ, we define the corresponding Kernel to be
$$K(x, z) = \phi(x)^T \phi(z).$$
- Then, everywhere we previously had $\langle x, z \rangle$ in our algorithm, we could simply replace it with $K(x, z)$, and our algorithm would now be learning using the features ϕ.
- Something interesting is that often, $K(x, z)$ may be very inexpensive to calculate, even though $\phi(x)$ itself may be very expensive to calculate.

Kernels

- In such settings, by using in our algorithm an efficient way to calculate $K(x, z)$, we can get SVMs to learn in the high dimensional feature space given by ϕ, but without ever having to explicitly find or represent vectors $\phi(x)$.
- Lets see an example. Suppose $x, z \in \mathbb{R}^n$, and consider

$$K(x, z) = (x^T z)^2.$$

- Note that computing $(x^T z)^2$ can be done in $O(n)$.

Kernels

We can also write this as

$$
\begin{aligned}
K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\
&= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)
\end{aligned}
$$

Thus, we see that $K(x,z) = \phi(x)^T \phi(z)$

Kernels

Where the feature mapping $\phi(x)$ is given by

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}.$$

Note that while calculating the high-dimensional $\phi(x)$ requires $O(n^2)$ time, finding $K(x, z)$ takes only $O(n)$ time.

Kernels

For a related kernel, consider

$$\begin{aligned} K(x,z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2. \end{aligned}$$

Kernels

This corresponds to the feature mapping

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix}.$$

Kernels

- More broadly, the kernel $K(x,z) = (x^T z + c)^d$ corresponds to a feature mapping to an $\binom{n+d}{d}$ feature space.
- This corresponds to all the monomials of the form$x_{i1}x_{i2}\ldots x_{ik}$ that are up to order d.
- However, despite working in dimension $O(n^d)$, computing $K(x,z)$, still takes only $O(n)$ time, and hence we never need to explicitly represent feature vectors in this very high dimensional space.

Kernels

- Intuitively, if $\phi(x)$ and $\phi(z)$ are close together, then we might expect $K(x,z) = \phi(x)^T\phi(z)$ to be large.
- Conversely, if $\phi(x)$ and $\phi(z)$ are far apart, then $K(x,z) = \phi(x)^T\phi(z)$ will be small.
- So we can think of $K(x,z)$ as some measurement of how similar are $\phi(x)$ and $\phi(z)$, or of how similar are x and z.

Kernels

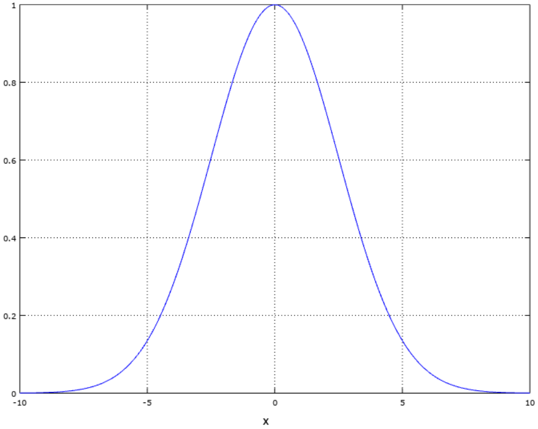
The Gaussian kernel corresponds to an infinite dimensional feature mapping ϕ, such that

$$K(x,z) = \exp\Big(-\frac{||x-z||^2}{2\sigma^2}\Big).$$

This is a reasonable measure of x and z similarity, and is close to 1 when x and z are close, and near 0 when x and z are far apart.

Kernels

Plot of the Gaussian kernel in one dimension with $x = 0$ and $\sigma = 0.2$:



Kernels

In a simple case with $x, z \in \mathbb{R}$ we have

$$
\begin{aligned}
K(x,z) &= \exp\Big(-\tfrac{||x-z||^2}{2\sigma^2} \Big) \\
&= \exp(-(x-z)^2) \\
&= \exp(-x^2)\exp(-z^2)\exp(2xz) \\
&= \exp(-x^2)\exp(-z^2)\sum_{k=0}^{\infty} \frac{2^k (x)^k (z)^k}{k!}
\end{aligned}
$$

Kernels

Given some function K, how can we tell if it is a valid kernel?

Mercer Theorem

Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then, for K to be a valid kernel, it is necessary and sufficient that for any $\{x^{(1)}, \dots, x^{(m)}\}, (m < \infty)$, the corresponding kernel matrix is symmetric and positive semi-definite.

where a kernel matrix $\mathcal{K}$ for a data set $\{x^{(1)}, \dots, x^{(m)}\}, (m < \infty)$ is defined as

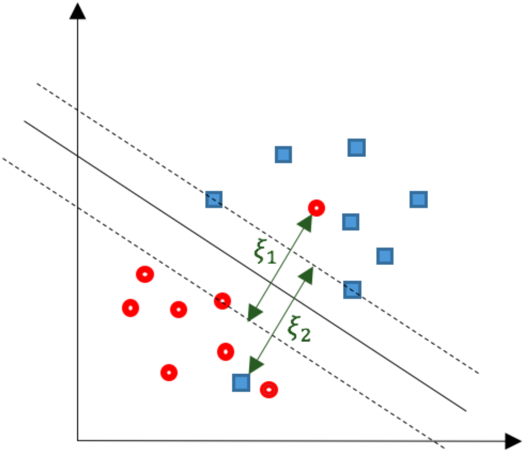
$$\mathcal{K} = \begin{bmatrix} K(x^{(1)}, x^{(1)}) & K(x^{(1)}, x^{(2)}) & \dots & K(x^{(1)}, x^{(m)}) \\ K(x^{(2)}, x^{(1)}) & K(x^{(2)}, x^{(2)}) & \dots & K(x^{(2)}, x^{(m)}) \\ \dots & \dots & \dots & \dots \\ K(x^{(m)}, x^{(1)}) & K(x^{(m)}, x^{(2)}) & \dots & K(x^{(m)}, x^{(m)}) \end{bmatrix}.$$

Kernels

The idea of kernels has significantly broader applicability than SVMs:

Any algorithm that can be written in terms of only the inner products $\langle x, z \rangle$ between input attribute vectors, then by replacing this with $K(x, z)$ where K is a kernel function, you can allow the algorithm to work efficiently in the high dimensional feature space corresponding to K.

The Non-Separable Case



The Non-Separable Case

To make the algorithm work for non-linearly separable datasets as well as less sensitive to outliers, we reformulate our optimization problem as follows:

$$\begin{aligned} \min_{\gamma,\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

Thus, examples are now allowed to have functional margin less than 1.

The Non-Separable Case

- Therefore, when an example i has a functional margin $1 - \xi_i$, the cost of that solution will be increased by $C\xi_i$.
- The parameter C controls the relative weighting between the twin goals of making $||\mathbf{w}||^2$ large and of ensuring that most examples have functional margin at least 1.

The Non-Separable Case

As before the Lagrangian is:

$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, r) =$

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\left[y^{(i)}(\mathbf{w}^T\mathbf{x} + b) - 1 + \xi_i\right] - \sum_{i=1}^{m} r_i\xi_i.$$

where α_i and r_i are Lagrange multipliers constrained to be ≥ 0.
After setting the derivatives with respect to $\mathbf{w}$ and b to zero we get:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned}$$

Sequential Minimal Optimization

- Consider trying to solve the unconstrained optimization problem

$$\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m).$$

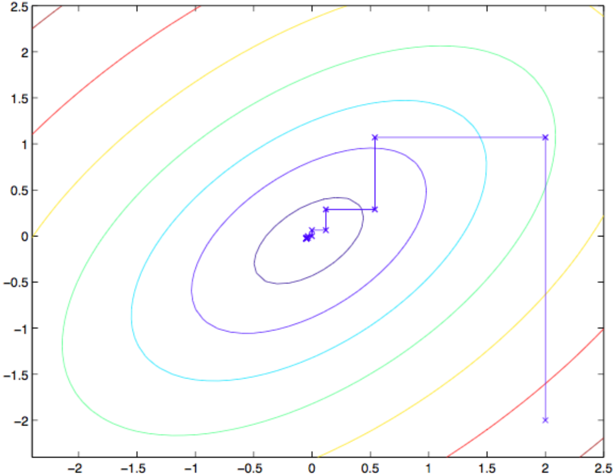
- We can use the coordinate ascent algorithm to solve it:

  Loop until convergence

  For $i = 1, \ldots, m$

$$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m).$$

Sequential Minimal Optimization



Sequential Minimal Optimization

From the optimization problem

$$
\begin{aligned}
\max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \\
\text{s.t.} \quad & 0 \leq \alpha_i \leq C, \;\; i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0
\end{aligned}
$$

we have that

$$\alpha_1 y^{(1)} = -\sum_{i=2}^{m} \alpha_i y^{(i)}.$$

this is

$$\alpha_1 = -y^{(1)} \sum_{i=2}^{m} \alpha_i y^{(i)}.$$

Sequential Minimal Optimization

Repeat until convergence

1. Select some pair α_i and α_j to update next.
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the others $\alpha_k,\ k \neq i, j$ fixed.

Sequential Minimal Optimization

From

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

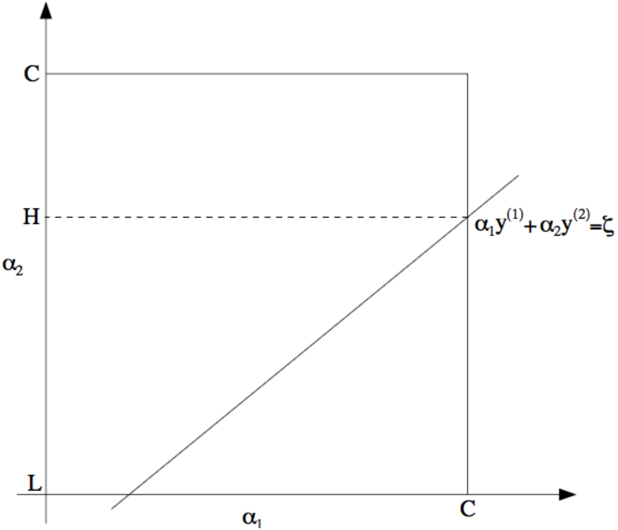
we can also see that

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)}.$$

and by making $-\sum_{i=3}^{m} \alpha_i y^{(i)} = \zeta$ a constant, we get

$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta.$ $\qquad \alpha_1 = \left(\zeta - \alpha_2 y^{(2)}\right) y^{(1)}.$

Sequential Minimal Optimization



Sequential Minimal Optimization

Writing our objective function $W(\alpha)$ as

$$W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W((\zeta - \alpha_2 y^{(2)})\, y^{(1)}, \alpha_2, \alpha_3, \ldots, \alpha_m)$$

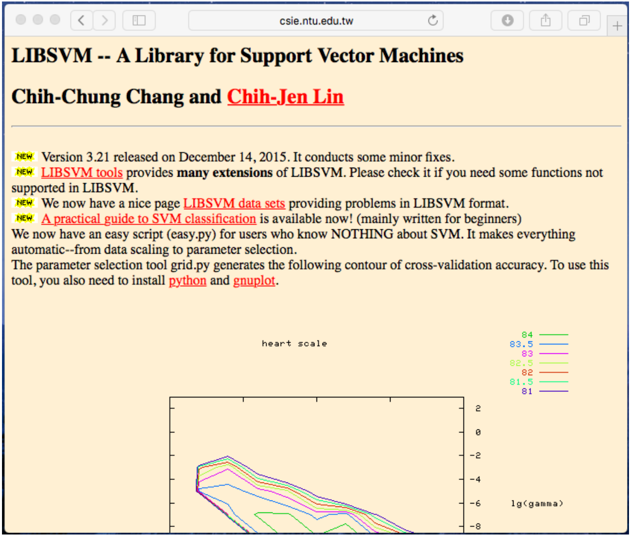
and treating $\alpha_3, \ldots, \alpha_m$ as constants, we get a quadratic function that can be easily maximized by setting its derivative to zero and solving for α_2.

If $\alpha_2 > H$, we make $\alpha_2 = H$.

If $\alpha_2 < L$, we make $\alpha_2 = L$.

Finally, having found α_2, we can calculate α_1 from $\alpha_1 = (\zeta - \alpha_2 y^{(2)})\, y^{(1)}$.

The libSVM library



Reference

- Andrew Ng. **Machine Learning Course Notes**. 2003.
- Christopher Bishop. **Pattern Recognition and Machine Learning**. Springer. 2006.

Thank You!

Dr. Víctor Uc Cetina
cetina@informatik.uni-hamburg.de