CS281B/Stat241B: Advanced Topics in Learning & Decision Making

The Kernel Trick

Lecturer: Michael I. Jordan *Scribe: Romain Thibaux*

1 Support Vectors

1.1 Solving the dual problem

Last time we transformed our original max-margin problem to its dual version:

$$\begin{aligned} \max_{\alpha \geq 0} \quad & \theta(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \end{aligned}$$

This is a quadratic program which we can solve using a number of techniques. The easiest (but innefficient) way is gradient projection. Or simply give it to Matlab. Obtaining the optimal α will give us the ω vector we really care about using the simple relation we showed earlier:

$$\omega = \sum_i \alpha_i y_i x_i$$

However originally we wanted ω **and** b because at the end of the day we want to classify new data points x_{new} by testing:

$$\omega^T x_{\text{new}} + b \geq 0 \quad ?$$

So how do we get b? To answer this we need to digress to introduce the **KKT conditions** for optimality (**K**arush **K**uhn **T**ucker).

1.2 The KKT conditions

Recall that for a general convex optimization problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0 \end{aligned}$$

we introduced the Lagrangian:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^T g(x)$$

where $\lambda^T g(x)$ acts as a force atracting x to the feasible set, away from the non-feasible set. At optimality, the solution satisfies the KKT conditions:

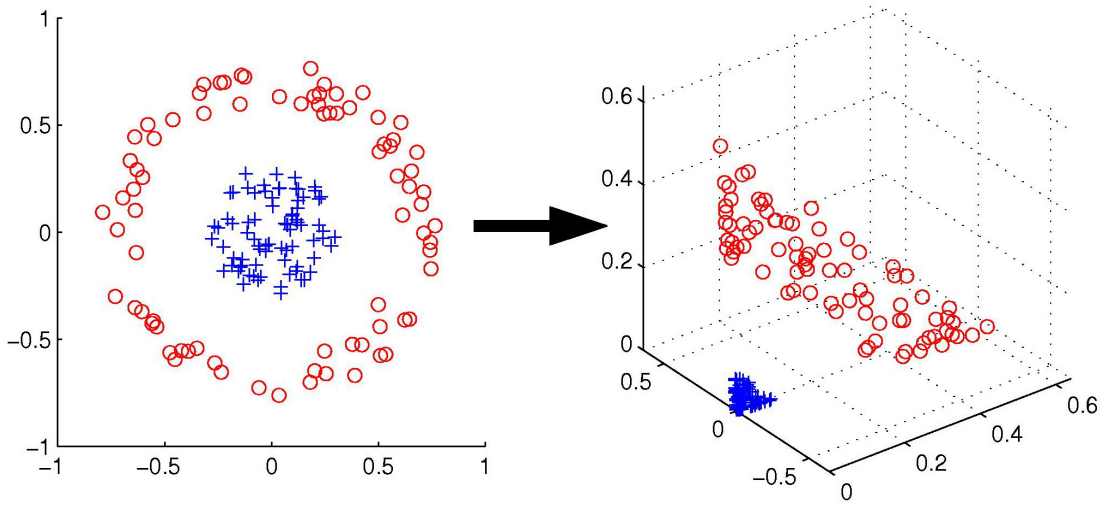

Figure 1: Transforming the data can make it linearly separable

- both x and λ are feasible
- $\forall i \ \lambda_i g(x_i) = 0$

These conditions make intuitive sense, in particular if we imagine a single constraint. The unconstrained optimum is either inside the feasible set, or outside. If it is inside, then the constrained and unconstrained optima are the same. x is atracted to this optimum, therefore there is no need to introduce a force to convince it to stay in the feasible set, so $\lambda_i = 0$. If it is outside, then x will want to escape the feasible set. We will need a non-zero λ_i to retain it, but x will go as far as possible towards the non-feasible set. Therefore it will be on the boundary, where $g(x_i) = 0$. So one of the two must be zero.

1.3 The Support Vectors

Why does this matter to us ? What do the KKT conditions imply in our case ?
In our case our Lagrange multiplier is α, and

$$g(x_i) = 1 - y_i(\omega^T x_i + b)$$

Now, at the optimum, pick an i such that $\alpha_i > 0$. Then the KKT conditions imply $g(x_i) = 0$ so:

$$\begin{aligned} & y_i(\omega^T x_i + b) = 1 \\ \text{i.e.} \quad & b = y_i - \omega^T x_i \end{aligned}$$

Any i such that $\alpha_i > 0$ will do, but in practice it is better (for numerical stability) to average the result obtained using several of them.
The corresponding x_i's are called the **support vectors** since they support the hyperplanes on both sides of the margin. And this is why the max-margin classifier is also called **support vector machine** (SVM). However when people refer to SVM, they generally refer to the enhanced and more general version that we will now describe.

2 The Kernel Trick

All the algorithms we have described so far use the data only through inner products. Because of this, they can be made non-linear in a very general way. Let's start by an example:

2.1 Example

Clearly, the data on the left in figure 1 is not linearly separable. Yet if we map it to a three-dimensional space using

$$\begin{array}{rccc} \phi: & \Re^2 & \longrightarrow & \Re^3 \\ & (x_1, x_2) & \longmapsto & (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array}$$

and if we try to linearly separate the mapped data, our decision boundaries will be hyperplanes in $\Re^3$, of the form $\omega^T z + b = 0$, i.e. as a function of x they will be of the form

$$\omega_1 x_1^2 + \omega_2\sqrt{2}x_1x_2 + \omega_3 x_2^2 = 0$$

which is the equation of an ellipse. So that's interesting, we can use our linear algorithm on a transformed version of the data to get a non-linear algorithm with no effort !
But look more closely at what the algorithm is doing. All we use is the Gram Matrix K of the data, in the sense that once we know K, we can throw away our original data.

$$K = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots \\ x_2^T x_1 & \ddots & \\ \vdots & & \end{bmatrix}_{n*n} = XX^T$$

$$\text{where} \qquad X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}_{n*d}$$

X, containing all the data, is called the **design matrix**.
What happens in our example when we first map our data via some function ϕ ? The Gram Matrix is now that of the z_i's:

$$K = \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & \cdots \\ \phi(x_2)^T\phi(x_1) & \ddots & \\ \vdots & & \end{bmatrix}$$

Let's write these inner products. Let r and s be vectors in $\Re^3$ corresponding to a and b respectively.

$$\begin{aligned} \langle r, s\rangle &= r_1s_1 + r_2s_2 + r_3s_3 \\ &= a_1^2b_1^2 + 2a_1a_2b_1b_2 + a_2^2b_2^2 \\ &= \langle a, b\rangle^2 \end{aligned}$$

So instead of mapping our data via ϕ and computing the inner product, we can do it in one operation, leaving the mapping completely implicit. In fact in the end we don't even need to know ϕ, all we need to know is how to compute the modified inner product. And because "modified inner product" is a long name, we will call it a **kernel**, $K(x, y)$. We will also call K the **kernel matrix** because it contains the value of the kernel for every pair of data points, thus using same letter both for the function and its matrix.
Because the kernel seems to be the object of interest, and not the mapping ϕ, we would like to characterize kernels without this mere intermediate. **Mercer's Theorem** gives us just that.

2.2 Mercer's Theorem

A symmetric function $K(x, y)$ can be expressed as an inner product

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

for some ϕ if and only if $K(x, y)$ is positive semidefinite, i.e.

$$\int K(x, y)g(x)g(y)dxdy \geq 0 \qquad \forall g$$

or, equivalently:

$$\begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ K(x_2, x_1) & \ddots & \\ \vdots & & \end{bmatrix} \text{ is psd for any collection } \{x_1 \dots x_n\}$$

Therefore you can either explicitly map the data with a ϕ and take the dot product, or you can take any kernel and use it right away, without knowing nor caring what ϕ looks like. For example:

- Gaussian Kernel: $K(x, y) = e^{\frac{1}{2}||x-y||^2}$
- Spectrum Kernel: count the number of substrings in common. It is a kernel since it is a dot product between vectors of indicators of all the substrings.