Linear Classification: The Kernel Method

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Recall the core problem of linear classification:

> Let P be a set of points in $\mathbb{R}^d$, each of which carries a label 1 or -1. The goal of the **linear classification problem** is to determine whether there is a d-dimensional plane
>
> $$x_1 \cdot c_1 + x_2 \cdot c_2 + ... + x_d \cdot c_d = 0$$
>
> which separates the points in P of the two labels.

If the plane exists, then P is said to be **linearly separable**. Otherwise, P is **linearly non-separable**.

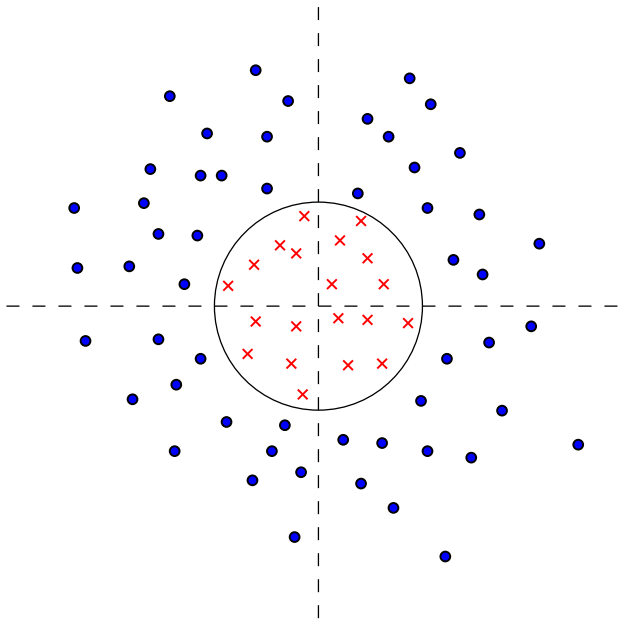
Why the Separable Case Is Important?

It seems that so far we have not paid much attention to non-separable datasets: indeed, all the techniques we have learned are designed for the scenario where P is linearly separable.

In this lecture, we will see that there is actually a good reason for this. We will learn a technique—called the kernel method—that maps a dataset to another (usually higher dimensional) data space, where coordinates are synthesized from those in the original space. The conversion significantly increases the chance that the (new) dataset is linearly separable. In fact, certain mappings even guarantee linear separability.

Motivation

Consider the non-separable circle dataset P below, where a point p has label 1 if $(p[1])^2 + (p[2])^2 \leq 1$, or -1 otherwise.

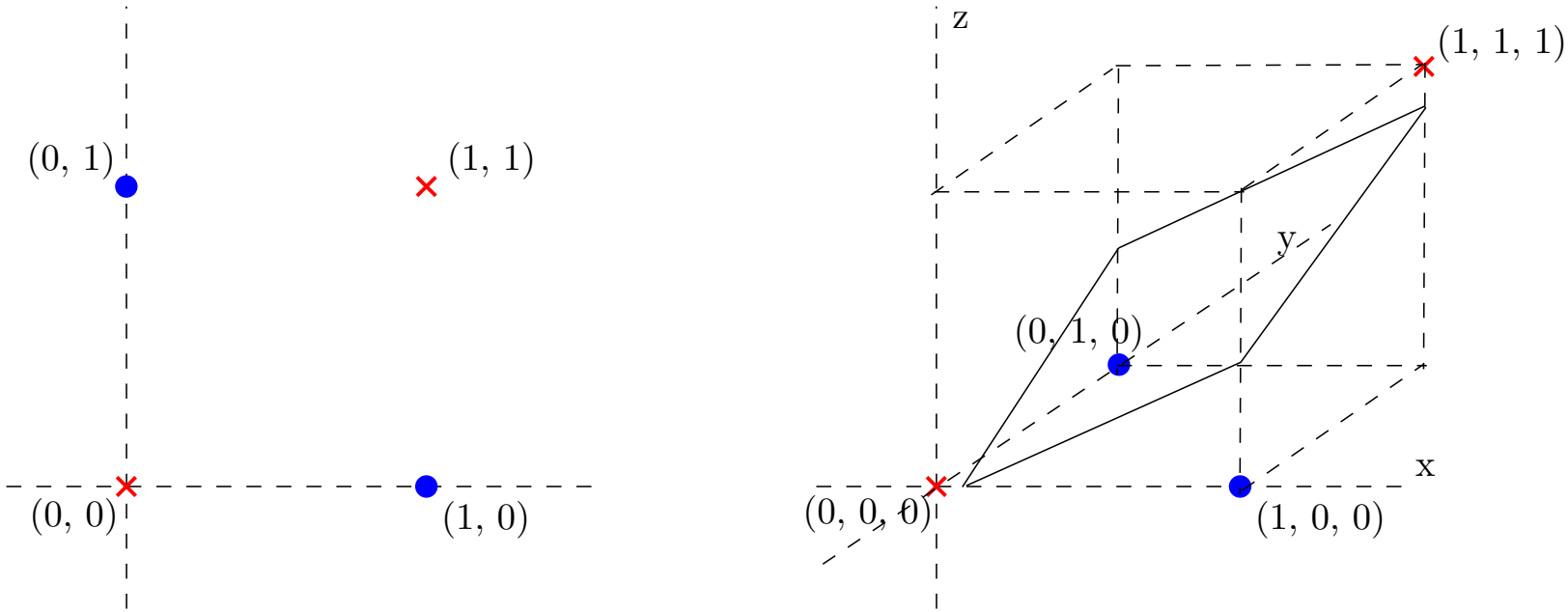


Let us map each point $p \in P$ to a point p' in another data space where $p'[1] = (p[1])^2$ and $p'[2] = (p[2])^2$. This gives a new dataset P'.

Clearly the points in P' of the two labels are separated by a linear plane $p'[1] + p'[2] = 1$.

Motivation

The left figure below is another non-separable dataset P (known as the XOR dataset).



The right figure shows the 4 points after the transformation from a 2D point (x, y) to a 3D point (x, y, xy). The new dataset is linearly separable.

Increasing the Dimensionality Guarantees Linearly Separability

Theorem: Let P be an arbitrary set of n points in 1D space, each of which has label 1 or -1. We can map each point $x \in P$ to an n-dimensional point $(1, x, x^2, ..., x^{n-1})$, such that the resulting point set becomes linearly separable.

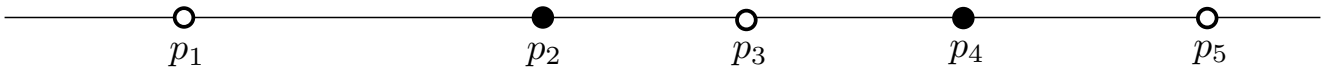
Think: Why does it suffice to focus on 1D? (Hint: for 2D, just take the x-coordinates; if there're duplicates, rotate the space).

We will prove the theorem in the next two slides.

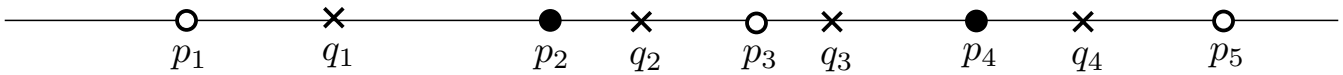
Increasing the Dimensionality Guarantees Linearly Separability

Proof: Denote the points in P as $p_1, p_2, ..., p_n$ in ascending order. We will consider that n is an odd number (the case where n is even is similar, and is left to you). Without loss of generality, assume that p_i has label -1 when $i \in [1, n]$ is an odd integer, and 1 otherwise.

The following shows an example where $n = 5$; white and black points have labels -1 and 1, respectively.



Between p_i and p_{i+1} $(1 \leq i \leq n-1)$, pick an arbitrary point q_i. The figure below shows an example:



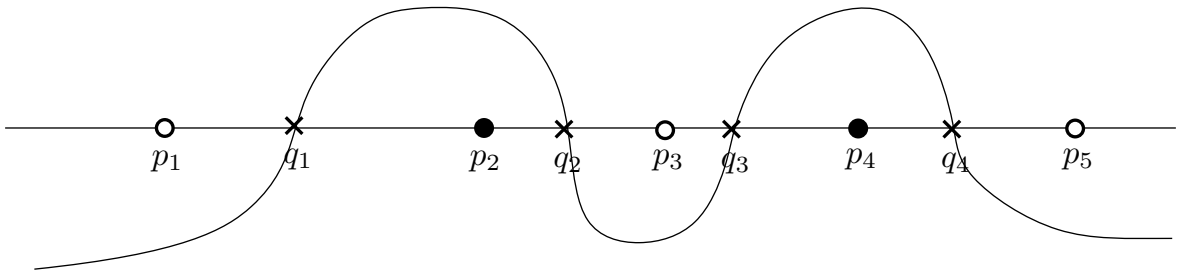
Increasing the Dimensionality Guarantees Linearly Separability

Proof (cont.): Now consider the following polynomial function

$$f(x) \quad = \quad -(x - q_1)(x - q_2)...(x - q_{n-1}).$$

It must hold that: for every label-(-1) point p, $f(p) < 0$, while for every label-1 point, $f(p) > 0$.

The figure below shows what happens when $n = 5$:



Increasing the Dimensionality Guarantees Linearly Separability

Proof (cont.): Function $f(x)$ can be expanded into the following form:

$$f(x) = c_0 + c_1 x + c_2 x^2 + ... + c_{n-1} x^{n-1}.$$

Therefore, if we convert each point $x \in P$ to a point $(1, x, x^2, ..., x^{n-1})$, the resulting set of n-dimensional points must be separable by a plane passing the origin (of the n-dimensional space). □

In the proof, we have assumed that the labels of the points are "interleaving" (i.e., 0 followed by 1 and then by 0, etc.). What if this is not the case?

Issues

The previous theorem gives us confidence that "converting data to a higher dimensional space" works. However, the conversion given in the lemma is unlikely to be a good one (it "overfits" the training data). There are two main issues to consider in practice:

- **Issue 1:** Given a non-separable P, how to do the conversion more "properly"?
- **Issue 2:** By converting a d-dimensional space to a d'-dimensional space, computation could become much more expensive. In fact, even enumerating all the coordinates of a d'-dimensional point takes $\Theta(d')$ time! When $d' \gg d$ (e.g., $d' = n$ in the previous proof), the overall learning time can be very expensive.

Issues (Remedies)

The kernel method provides a good remedy to the above issues:

For Issue 1:

- Convert the data to a "sufficiently" high dimensionality.
- If one conversion does not work, try another one.

For Issue 2:

- Clever math tricks allow us to compute dot products in far less than $O(d')$ time even when $d' \gg d$.

Kernel Function

A **kernel function** K is a function from $\mathbb{R}^d \times \mathbb{R}^d$ to $\mathbb{R}$ with the following property: there is a mapping $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that, given any two points $p, q \in \mathbb{R}^d$, $K(p, q)$ equals the dot product of $\phi(p)$ and $\phi(q)$.

We will refer to the space $\mathbb{R}^{d'}$ (where $\phi(p)$ is) as the **feature space**.

We will see two common kernel functions next. Henceforth, a point $p = (p[1], p[2], ..., p[d])$ in $\mathbb{R}^d$ will interchangeably be regarded as a vector. For example, the dot product of two points p, q—written as $p \cdot q$—equals $\sum_{i=1}^{d} p[i]q[i]$.

Polynomial Kernel

Let p and q be two points in $\mathbb{R}^d$. A **polynomial Kernel** has the form:

$$K(p, q) \quad = \quad (p \cdot q + 1)^c$$

for some integer degree $c \geq 1$.

Example

Consider that $d = 2$ and $c = 2$. We can expand the Kernel function as:

$$\begin{aligned} K(p,q) &= (p \cdot q + 1)^2 = (p[1]q[1] + p[2]q[2] + 1)^2 \\ &= 1 + (p[1])^2(q[1])^2 + (p[2])^2(q[2])^2 + \\ &\quad 2(p[1]p[2])(q[1]q[2]) + 2p[1]q[1] + 2p[2]q[2]. \end{aligned}$$

We can regard the above as the dot product of $\phi(p)$ and $\phi(q)$, where $\phi(p)$ is a 6 dimensional point:

$$\phi(p) = (1, p[1]^2, p[2]^2, \sqrt{2}p[1]p[2], \sqrt{2}p[1], \sqrt{2}p[2]).$$

In other words, the converted data space has a dimensionality of $d' = 6$.

Note that the Kernel function is powerful enough to make both the circle dataset and the XOR dataset linearly separable!

In general, a polynomial Kernel with degree c converts d-dimensional space to $\binom{d+c}{c}$ dimensional space.

Gaussian Kernel (a.k.a. RBF Kernel)

Let p and q be two points in $\mathbb{R}^d$. A **Gaussian Kernel** has the form:

$$K(p,q) \quad = \quad \exp\left(-\frac{\|p,q\|^2}{2\sigma^2}\right)$$

for real value $\sigma > 0$. Recall that $\|p,q\|$ is the Euclidean distance between p and q, namely, $\|p,q\|^2 = \sum_{i=1}^{d}(p[i]-q[i])^2$.

In general, a Gaussian Kernel converts d-dimensional space to another space with infinite dimensionality! We will illustrate this in the next slide for $d = 1$.

Gaussian Kernel (a.k.a. RBF Kernel)

We know from Taylor expansion:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ...$$

When $d = 1$, $\|p - q\|^2 = (p^2 - 2pq + q^2)$. Hence:

$$\exp\left(-\frac{\|p,q\|^2}{2\sigma^2}\right) = \exp\left(-\frac{p^2 - 2pq + q^2}{2\sigma^2}\right) = \exp\left(-\frac{p^2 + q^2}{2\sigma^2}\right)\exp(\frac{pq}{\sigma^2})$$

$$= \frac{1}{e^{\frac{p^2}{2\sigma^2}}}\frac{1}{e^{\frac{q^2}{2\sigma^2}}}\exp(\frac{pq}{\sigma^2})$$

$$= \frac{1}{e^{\frac{p^2}{2\sigma^2}}}\frac{1}{e^{\frac{q^2}{2\sigma^2}}}\left(1 + \frac{pq}{\sigma^2} + \frac{(p/\sigma)^2(q/\sigma)^2}{2!} + \frac{(p/\sigma)^3(q/\sigma)^3}{3!} + ...\right)$$

It is now clear that $\phi(p)$ has the following coordinates:

$$\left(\frac{1}{e^{\frac{p^2}{2\sigma^2}}}, \frac{p/\sigma}{e^{\frac{p^2}{2\sigma^2}}}, \frac{(p/\sigma)^2}{\sqrt{2!}\cdot e^{\frac{p^2}{2\sigma^2}}}, \frac{(p/\sigma)^3}{\sqrt{3!}\cdot e^{\frac{p^2}{2\sigma^2}}}, ...\right)$$

Finding a Separation Plane in the Converted Space

We have seen that, using a Kernel function $K(.,.)$, we have converted the original d-dimensional dataset P into another d'-dimensional dataset $P' = \{\phi(p) \mid p \in P\}$ where typically $d' \gg d$. Very likely P' is linearly separable. But how do we find a separation plane in the converted space $\mathbb{R}^{d'}$?

One (naive) idea is to materialize P', but this requires figuring out the details of $\phi(.)$. As shown earlier, this is either cumbersome (e.g., polynomial kernel) or impossible (e.g., Gaussian Kernel).

It turns out that we can achieve the purpose without working in the d'-dimensional space at all. Our weapon is, once again, Perceptron! We will utilize the fact that the entire algorithm can be implemented using only the Kernel function. This allows us to work directly in the original d-dimensional space.

Recall:

Perceptron

The algorithm starts with $c = (0, 0, ..., 0)$, and then runs in iterations.

In each iteration, it simply checks whether any point in $p \in P$ violates our requirement according to c. If so, the algorithm adjusts c as follows:

- If p has label 1, then $c \leftarrow c + p$.
- If p has label -1, then $c \leftarrow c - p$.

As soon as c has been adjusted, the current iteration finishes; and a new iteration starts.

The algorithm finishes if the iteration finds all points of P on the right side of the plane.

In the converted space $\mathbb{R}^{d'}$, it should be modified as:

Perceptron

The algorithm starts with $c = (\underbrace{0, 0, ..., 0}_{d'})$, and then runs in iterations.

In each iteration, it simply checks whether any point in $\phi(p) \in P'$ violates our requirement according to c. If so, the algorithm adjusts c as follows:

- If $\phi(p)$ has label 1, then $c \leftarrow c + \phi(p)$.
- If $\phi(p)$ has label -1, then $c \leftarrow c - \phi(p)$.

As soon as c has been adjusted, the current iteration finishes; and a new iteration starts.

The algorithm finishes if the iteration finds all points of P' on the right side of the plane.

Next we will show how to implement the algorithm using the Kernel function $K(.,.)$.

Perceptron

For point $p \in P$, denote by t_p the number of times that p has been used to adjust c ($t_p = 0$ if p has never been used before). Let P_{-1} (or P_1) be the set of label-(-1) (or label-1, resp.) points in P.

Hence, the current c is:

$$c = \sum_{p \in P_{-1}} t_p \phi(p) - \sum_{p \in P_1} t_p \phi(p).$$

Perceptron

The key step to implement is this: given an arbitrary point $q \in \mathbb{R}^d$, we want to compute the dot product between c and $\phi(q)$ in the d'-dimensional space. Using the Kernel function $K(.,.)$, we have:

$$\begin{aligned} c \cdot \phi(q) &= \left(\sum_{p \in P_{-1}} t_p \phi(p) - \sum_{p \in P_1} t_p \phi(p) \right) \phi(q) \\ &= \left(\sum_{p \in P_{-1}} t_p (\phi(p) \cdot \phi(q)) \right) - \left(\sum_{p \in P_1} t_p (\phi(p) \cdot \phi(q)) \right) \\ &= \sum_{p \in P_{-1}} t_p \cdot K(p,q) - \sum_{p \in P_1} t_p \cdot K(p,q). \end{aligned}$$

Therefore, by maintaining t_p for every $p \in P$, we never need to compute any dot-products in the converted d'-dimensional space.

In summary, the kernel method allows us to convert the original d-dimensional space into another d'-dimensional space with $d' > d$. In practice, usually it is a good idea to keep d' as low as possible. Otherwise, we may be **overfitting** the training set, i.e., producing a linear classifier that works too well on the training data, but not on the unseen data.

Rules of thumb:

- Polynomial kernels with small degrees are often good choices.
- Gaussian kernel is also a good choice when the R^2/γ^2 is small in the converted (infinitely-dimensional) space.
  - Both R and γ can be computed using Kernel functions (think: how).