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

$$x_1 \cdot c_1 + x_2 \cdot c_2 + \ldots + x_d \cdot c_d = 0$$

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?

So far, we have not paid much attention to non-separable datasets. All the techniques we have learned are designed for the scenario where $P$ is linearly separable.

This lecture will give a good reason for this. We will learn a technique — called the kernel method — that maps a dataset into another space of higher dimensionality. By applying the method appropriately, we can always guarantee linear separability.
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 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.
**Theorem:** Let $P$ be an arbitrary set of $n$ points in 1D space, each of which has label 1 or $-1$. If we map each point $x \in P$ to an $n$-dimensional point $(1, x, x^2, ..., x^{n-1})$, the set of points obtained is always linearly separable.

**Think:** How do you apply the result in 2D? (Hint: just take the $x$-coordinates; if there are 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 $p_1, p_2, \ldots, p_n$ in ascending order. We will consider that $n$ is an odd number (the opposite case 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.

Here, the labels of the points are “interleaving” (i.e., $-1, 1, -1, 1, \ldots$). After you have understood the proof, think how to extend it a non-interleaving $P$.

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

```
| p_1 | p_2 | p_3 | p_4 | p_5 |
```

Increasing the Dimensionality Guarantees Linearly Separability

**Proof (cont.):** 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:

![Diagram showing linearly separable points]

Now consider the following polynomial function

\[
f(x) = -(x - q_1)(x - q_2)\ldots(x - q_{n-1}).
\]

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 \( n = 5 \):

![Diagram showing the polynomial function and its effect on the points]
Increasing the Dimensionality Guarantees Linearly Separability

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

$$f(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{n-1}x^{n-1}.$$  

Therefore, if we convert each point $x \in P$ to a point $(1, x, x^2, \ldots, x^{n-1})$, the resulting set of $n$-dimensional points must be separable by a plane passing the origin (of the $n$-dimensional space).
The conversion explained in the proof produces a new space of dimensionality $d' = n$. When $d'$ is large, computation in the converted space can be very expensive (in fact, even enumerating all the coordinates of point takes $\Theta(d')$ time). Is it possible improve the efficiency?

This is where kernel functions come into the picture.
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 $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ 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 **kernel 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 $p$. For example, the dot product of two points $p, q$ — written as $p \cdot q$ — equals $\sum_{i=1}^{d} p[i]q[i]$. 
Let $p$ and $q$ be two points in $\mathbb{R}^d$. A **polynomial kernel** has the form:

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

for some integer degree $c \geq 1$. 
Consider that $d = 2$ and $c = 2$. We can expand the Kernel function as:


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


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

**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:

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

for a real value $\sigma > 0$ called the **bandwidth**. Note that $\text{dist}(p, q)$ is the Euclidean distance between $p$ and $q$, namely,

$$\text{dist}(p, q)^2 = \sum_{i=1}^{d}(p[i] - q[i])^2.$$

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

---

**Y Tao**

Linear Classification: The Kernel Method
Gaussian Kernel (a.k.a. RBF Kernel)

We know from Taylor expansion $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$

When $d = 1$, $\text{dist}(p, q)^2 = p^2 - 2pq + q^2$. Hence:

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

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

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

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

$$
\left( \frac{1}{e^{\frac{p^2}{2\sigma^2}}}, \frac{p/\sigma}{e^{\frac{p^2}{2\sigma^2}}}, \frac{(p/\sigma)^2}{2! e^{\frac{p^2}{2\sigma^2}}}, \frac{(p/\sigma)^3}{3! e^{\frac{p^2}{2\sigma^2}}}, \ldots \right)
$$
**Gaussian Kernel (a.k.a. RBF Kernel)**

**Theorem:** Regardless of the choice of $\sigma$, a Gaussian kernel is capable of separating any finite set of points.

The proof will be left as an exercise (with hints).
Finding a Separation Plane in the Converted Space

A Kernel function $K(.,.)$ allows us to convert the original $d$-dimensional dataset $P$ into another $d'$-dimensional dataset $P' = \{\phi(p) | p \in P\}$ where typically $d' \gg d$. But how do we find a separation plane in the kernel 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!
Recall:  

**Perceptron**

The algorithm starts with \( \mathbf{w} = (0, 0, \ldots, 0) \), and then runs in iterations.

In each iteration, it checks whether any point in \( p \in P \) violates our requirement according to \( \mathbf{w} \). If so, the algorithm adjusts \( \mathbf{w} \) as follows:

- If \( p \) has label 1, then \( \mathbf{w} \leftarrow \mathbf{w} + p \).
- If \( p \) has label \(-1\), then \( \mathbf{w} \leftarrow \mathbf{w} - p \).

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 $\mathbf{w} = (0, 0, ..., 0)$, and then runs in iterations. In each iteration, it simply checks whether any point in $\phi(p) \in P'$ violates our requirement according to $\mathbf{w}$. If so, the algorithm adjusts $\mathbf{w}$ as follows:

- If $\phi(p)$ has label 1, then $\mathbf{w} \leftarrow \mathbf{w} + \phi(p)$.
- If $\phi(p)$ has label $-1$, then $\mathbf{w} \leftarrow \mathbf{w} - \phi(p)$.

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 $K(.,.)$. 
For point $p \in P$, denote by $t_p$ the number of times that $p$ has been used to adjust $w$ ($t_p = 0$ 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 $w$ is:

$$w = \sum_{p \in P_1} t_p \phi(p) - \sum_{p \in P_{-1}} t_p \phi(p).$$
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 \( \mathbf{w} \) and \( \phi(q) \) in the \( d' \)-dimensional space. Using the Kernel function \( K(.,.) \), we have:

\[
\mathbf{w} \cdot \phi(q) = \left( \sum_{p \in P_1} t_p \phi(p) - \sum_{p \in P_{-1}} t_p \phi(p) \right) \phi(q) \\
= \left( \sum_{p \in P_1} t_p (\phi(p) \cdot \phi(q)) \right) - \left( \sum_{p \in P_{-1}} t_p (\phi(p) \cdot \phi(q)) \right) \\
= \sum_{p \in P_1} t_p \cdot K(p, q) - \sum_{p \in P_{-1}} t_p \cdot K(p, q).
\]

Therefore, by maintaining \( t_p \) for every \( p \in P \), we never need to compute any dot-products in the converted \( d' \)-dimensional space.
We finish this lecture with a question for you:

Think: How to apply the margin-based generalization theorem on the set $P'$ of points obtained by the kernel method?