

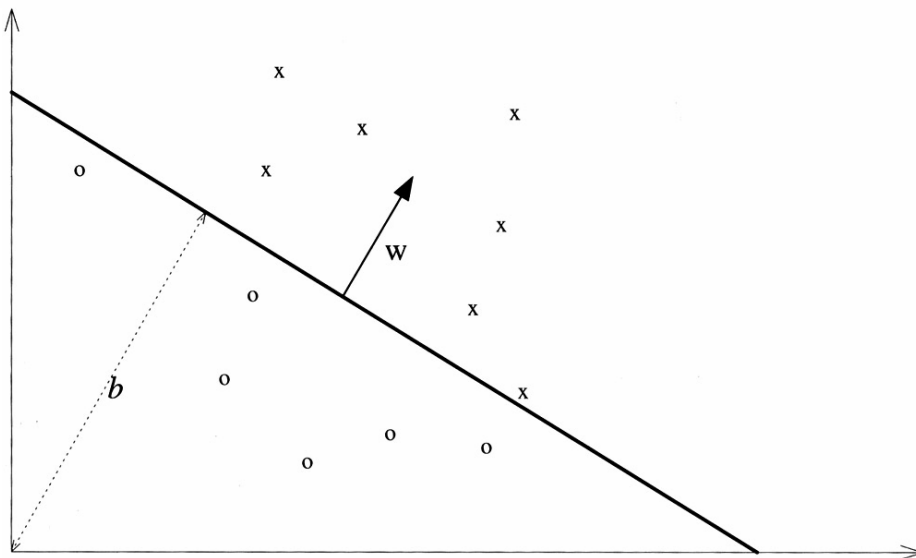
Kernel Methods

Linear Learning Machines

Binary classification is frequently performed by using a real-valued function $f : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ in the following way: the input $\mathbf{x} = (x_1, \dots, x_n)'$ is assigned to the positive class, if $f(\mathbf{x}) \geq 0$, and otherwise to the negative class. We consider the case where $f(\mathbf{x})$ is a linear function of $\mathbf{x} \in X$, so that it can be written as

$$\begin{aligned} f(\mathbf{x}) &= \langle \mathbf{w} \cdot \mathbf{x} \rangle + b \\ &= \sum_{i=1}^n w_i x_i + b \end{aligned}$$

where $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters that control the function and the decision rule is given by $\text{sgn}(f(\mathbf{x}))$, where we will use the convention that $\text{sgn}(0) = 1$. The learning methodology implies that these parameters must be learned from the data.



A separating hyperplane (\mathbf{w}, b) for a two dimensional training set

Rosenblatt's Perceptron

Given a linearly separable training set S and learning rate $\eta \in \mathbb{R}^+$

$\mathbf{w}_0 \leftarrow \mathbf{0}$; $b_0 \leftarrow 0$; $k \leftarrow 0$

$R \leftarrow \max_{1 \leq i \leq \ell} \|\mathbf{X}_i\|$

```
repeat
  for  $i = 1$  to  $\ell$ 
    if  $y_i(\langle \mathbf{w}_k, \mathbf{x}_i \rangle + b_k) \leq 0$  then
       $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$ 
       $b_{k+1} \leftarrow b_k + \eta y_i R^2$ 
       $k \leftarrow k + 1$ 
    end if
  end for
```

until no mistakes made within the *for* loop

return (\mathbf{w}_k, b_k) where k is the number of mistakes

Dual Representation

the final hypothesis will be a linear combination of the training points:

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{X}_i,$$

where, since the sign of the coefficient of \mathbf{x}_i is given by the classification y_i , the α_i are positive values proportional to the number of times misclassification of \mathbf{x}_i has caused the weight to be updated.

$$\begin{aligned}
h(\mathbf{x}) &= \text{sgn}(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b) \\
&= \text{sgn}\left(\left\langle \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j \cdot \mathbf{x} \right\rangle + b\right) \\
&= \text{sgn}\left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x} \rangle + b\right),
\end{aligned}$$

Given training set S

$\alpha \leftarrow \mathbf{0}; b \leftarrow 0$

$R \leftarrow \max_{1 \leq i \leq \ell} \|\mathbf{x}_i\|$

repeat

for $i = 1$ to ℓ

if $y_i \left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b \right) \leq 0$ **then**

$\alpha_i \leftarrow \alpha_i + 1$

$b \leftarrow b + y_i R^2$

end if

end for

until no mistakes made within the *for* loop

The Perceptron Algorithm (dual form)

Limitation of LLMs

Linear classifiers cannot deal with

- Non-linearly separable data
- Noisy data

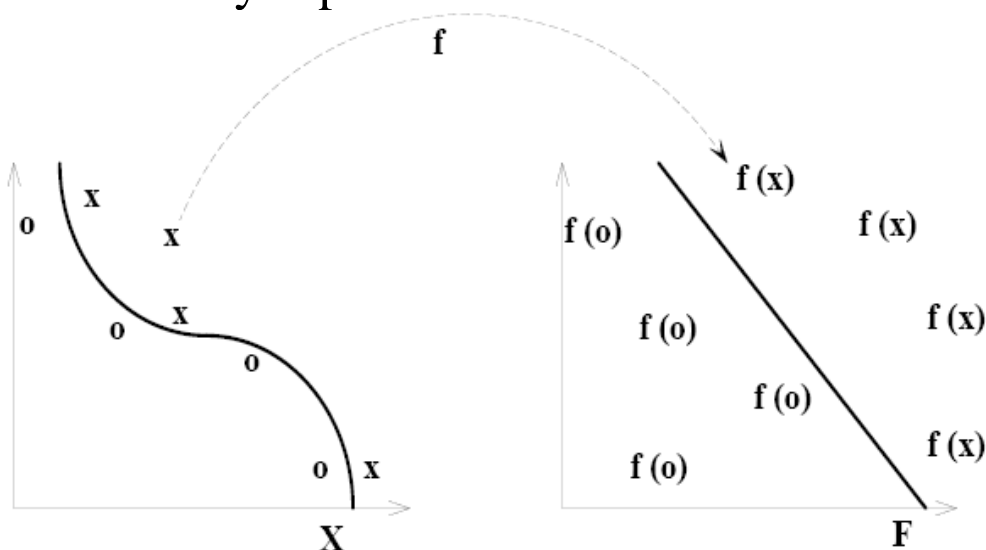
Also this formulation only deals with vectorial data

Non Linear Classifiers

- One solution: creating a net of simple linear Classifiers (neurons): Neural Networks
- Other solution: map data into a richer feature space including non-linear features, then use a linear classifier

Learning in the Feature Space

- Map data into a feature space where they are Linearly separable



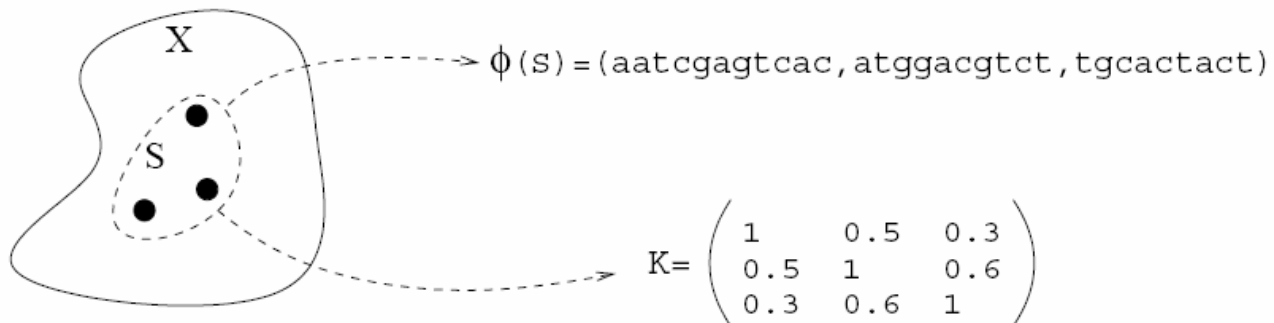
$$x \rightarrow \phi(x)$$

Problem with Feature Space

- Working in high dimensional feature spaces solves the problem of expressing complex functions
- **BUT: There is a computational problem (working with very large vectors)**

Kernel Representation

Kernel methods are based on a radically different answer to the question of data representation. Data are not represented individually anymore, but only through a set of pairwise comparisons. In other words, instead of using a mapping $\phi : \mathcal{X} \rightarrow \mathcal{F}$ to represent each object $\mathbf{x} \in \mathcal{X}$ by $\phi(\mathbf{x}) \in \mathcal{F}$, a real-valued “comparison function” $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is used, and the data set \mathcal{S} is represented by the $n \times n$ matrix of pairwise comparisons $k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. All kernel methods are designed to process such square matrices.



Two different representations of the same dataset. \mathcal{X} is supposed to be the set of all oligonucleotides, and \mathcal{S} is a data set of three particular oligonucleotides. The classic way to represent \mathcal{S} is first to define a representation $\phi(\mathbf{x})$ for each element of $\mathbf{x} \in \mathcal{X}$, for example, as a sequence of letters to represent the succession of nucleotides, and then to represent \mathcal{S} as the set $\phi(\mathcal{S})$ of representations of its elements (*upper part*). Kernel methods are based on a different representation of \mathcal{S} , as a matrix of pairwise similarity between its elements (*lower part*).

Comments

1. The representation as a square matrix does not depend on the nature of the objects to be analyzed. They can be images, molecules, or sequences, and the representation of a data set is always a real-valued square matrix. This suggests that an algorithm developed to process such a matrix can analyze images as well as molecules or sequences, as long as valid functions \mathbf{k} can be defined.
2. The size of the matrix used to represent a dataset of n objects is always $n * n$, whatever the nature or the complexity of the objects. For example, a set of ten tissues, each characterized by thousands of gene expression levels, is represented by a $10*10$ matrix, whatever the number of genes.

3. There are many cases where comparing objects is an easier task than finding an explicit representation for each object that a given algorithm can process. As an example, many data analysis algorithms, such as least squares regression or neural networks, require an explicit representation of each object \mathbf{x} as a vector $\phi(\mathbf{x}) \in \mathbb{R}^p$.

Gram Matrix

$\mathbf{K} =$

K(1,1)	K(1,2)	K(1,3)	...	K(1,m)
K(2,1)	K(2,2)	K(2,3)	...	K(2,m)
...
K(m,1)	K(m,2)	K(m,3)	...	K(m,m)

Kernels as Inner Products

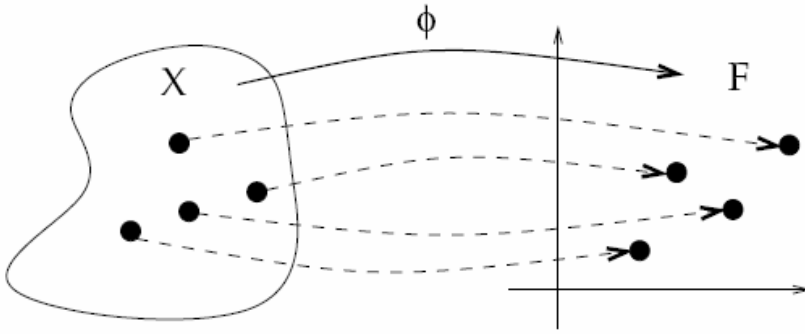
Suppose the data to be analyzed are real vectors, that is, $\mathcal{X} = \mathbb{R}^p$ and any object is written as $\mathbf{x} = (x_1, \dots, x_p)^\top$. One is tempted to compare such vectors using their inner product: for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$,

$$k_L(\mathbf{x}, \mathbf{x}') := \mathbf{x}^\top \mathbf{x}' = \sum_{i=1}^p x_i x'_i.$$

Theorem For any kernel k on a space \mathcal{X} , there exists a Hilbert space \mathcal{F} and a mapping $\phi : \mathcal{X} \rightarrow \mathcal{F}$ such that

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle, \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathcal{X},$$

where $\langle u, v \rangle$ represents the dot product in the Hilbert space between any two points $u, v \in \mathcal{F}$.



Any kernel on a space \mathcal{X} can be represented as an inner product after the space \mathcal{X} is mapped to a Hilbert space \mathcal{F} , called the feature space.

$$K(x, z) = \langle x, z \rangle^d$$

$$K(x, z) = e^{-\|x-z\|^2/2\sigma}$$

$$x = (x_1, x_2);$$

$$z = (z_1, z_2);$$

$$\langle x, z \rangle^2 = (x_1 z_1 + x_2 z_2)^2 =$$

$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 x_2 z_2 =$$

$$= \langle (x_1^2, x_2^2, \sqrt{2}x_1 x_2), (z_1^2, z_2^2, \sqrt{2}z_1 z_2) \rangle = \langle \phi(x), \phi(z) \rangle$$

This result, provides a first useful intuition about kernels: they can all be thought of as dot products in some space \mathcal{F} , usually called the *feature space*. Hence, using a kernel boils down to representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \mathcal{F}$, and computing dot products. There is, however, an important difference with respect to the explicit representation of objects as vectors, here the representation $\phi(\mathbf{x})$ does not need to be computed explicitly for each point in the data set \mathcal{S} , since only the pairwise dot products are necessary. In fact, there are many cases where the feature space associated with a simple kernel is infinite-dimensional, and the image $\phi(\mathbf{x})$ of a point \mathbf{x} is tricky to represent even though the kernel is simple to compute.