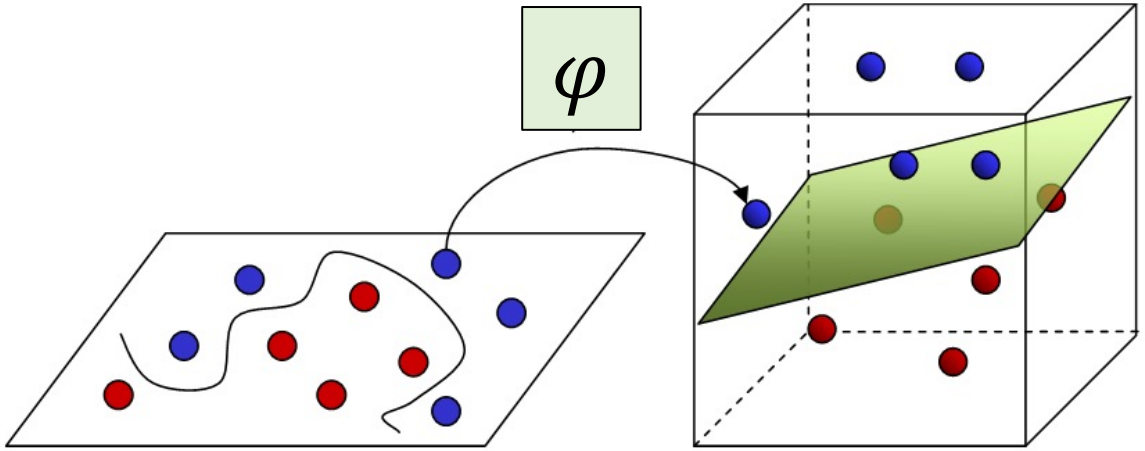
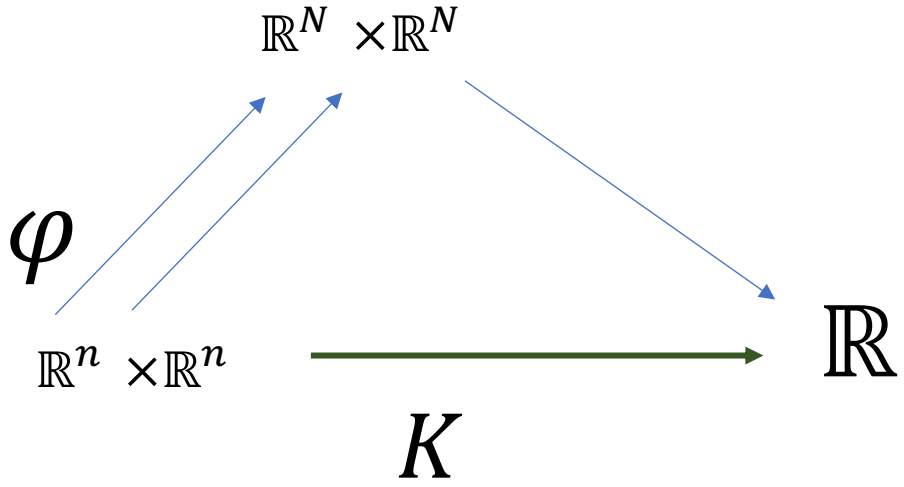
The Kernel Perceptron



Ariel Shamir

Zohar Yakhini

IDC HERZLIYA



Reminder: a systematic approach to classification by mapping into higher dimension

- Try to map into the full rational variety of increasing degrees.
- Apply the Perceptron in the mapping (ambient) space.

- Overfitting?
- The Perceptron uses inner products. What would the time complexity of the operation be?

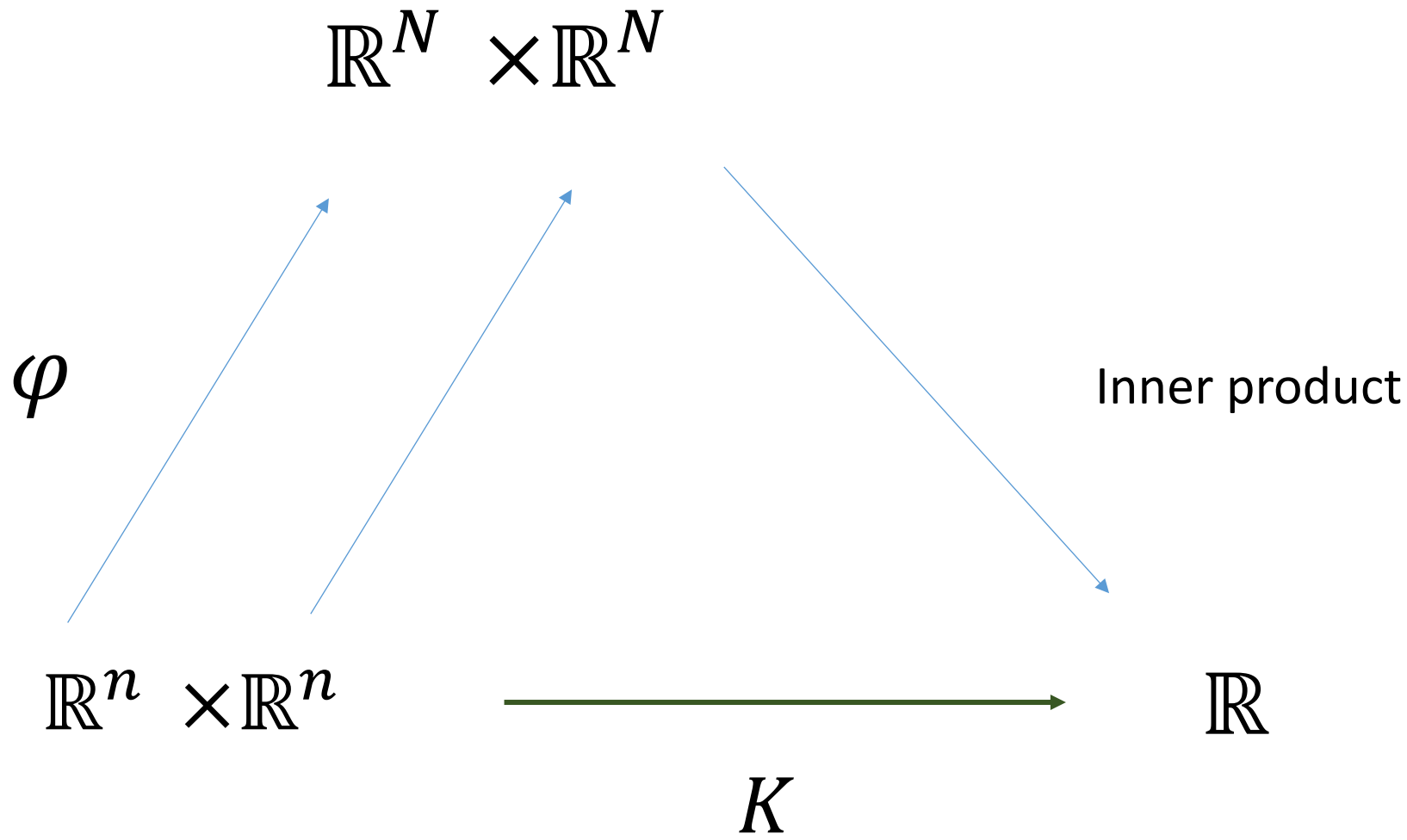
Here come the kernels …

Kernels

- A function $K: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ is called a **kernel** if there exists a mapping function $\varphi: \mathbb{R}^n \rightarrow \mathbb{R}^N$ so that the following always holds:

$$\forall x, y \quad K(x, y) = \varphi(x) \cdot \varphi(y)$$

- A mapping function $\varphi: \mathbb{R}^n \rightarrow \mathbb{R}^N$ is said to **afford a kernel** if such K exists.
  We are interested in the cases where φ affords an efficient kernel.
- Kernels transform the learning into direct operations in the (lower dimension) input space (will see how).
- Kernels help us avoid the explicit search for an ambient space and for mapping functions, φ, into higher dimensions. We explore kernels instead.
- In fact, kernels can also support learning in infinite dimensional mapping spaces (general Hilbert spaces)


$\mathbb{R}^N \times \mathbb{R}^N$
φ
Inner product
$\mathbb{R}^n \times \mathbb{R}^n$
$\mathbb{R}$
K

More Kernel Examples

$$\text{Homogenous Polynomial kernel}: \quad k(x,y)=(x\cdot y)^d$$

$$\text{Inhomogenous Polynomial kernel}: \quad k(x,y)=(x\cdot y+1)^d$$

$$\text{Radial Basis function kernel}: k(x,y)=e^{-\frac{\|x-y\|^2}{2\sigma^2}}$$

$$\text{Sums and products of kernels are kernels as well!}$$

Back to Rosenblatt's Perceptron

Learning:

Initialize each w_i to some small random number
Until termination do
For each $x_d \in D$ compute
$o_d = \text{sgn}(w \cdot x_d)$
For each linear weight w_i, do
$\Delta w_i = -\eta(o_d - t_d)(x_d)_i$
$w_i = w_i + \Delta w_i$

The computationally expensive step, when occurring in high dimension

We add a small part of x_d if it is misclassified

Classification, execution alg:

$$f(x) = \text{sgn}\left(\sum_i w_i x_i\right) = \text{sgn}(w \cdot x)$$

Closer look: Updating Weights

$$\Delta w_i = -\eta(o_d - t_d)x_{id}$$

1. Only misclassified examples affect the evolution of w.
2. If the output is **negative** and the target is **positive** then we add a **positive** fraction of the instance
3. If the output is **positive** and the target is **negative** then we add a **negative** fraction of the instance
4. Hence, in updating w,
   we always add a **positive fraction of** $t_d \mathbf{x}_d$
   (if its misclassified)

Recasting the decision function

$$C(x) = \text{sgn}\left(f(x)\right) = \text{sgn}\left(\sum_i w_i x_i\right) = \text{sgn}(w \cdot x)$$

$\vec{w}$ is a sum of $\eta t_d \vec{x}_d$ fractions, added in the learning cycles

$$f(x) = \vec{w} \cdot \vec{x} = \left(\sum_{d \in D} a_d t_d \vec{x}_d\right) \cdot \vec{x} = \sum_{d \in D} a_d t_d \left(\vec{x}_d \cdot \vec{x}\right)$$

There exist (non negative) numbers a_d , $d \in D$ that can be used to recast the decision function

Instance Based Learning/Execution

- Since w is a combination of fractions of the instances we had written the decision function as:

$$f(x) = \sum_d a_d \, t_d (\vec{x}_d \cdot \vec{x})$$

- Note that in order to use the decision function

$$f(x) = \sum_d a_d \, t_d (\vec{x}_d \cdot \vec{x})$$

we only need to store those data points $\vec{x}_d$ whose $a_d \neq 0$

- These are called "support vectors"

The Dual Perceptron

Initialize each w_i to some small random number.

Until termination do

For each $\vec{x}_d$ in D compute

$o_d = \text{sgn}(\vec{w} \cdot \vec{x}_d)$

For each linear unit weight w_i, Do

$\Delta w_i = -\eta(o_d - t_d)x_{id}$

$w_i = w_i + \Delta w_i$

Updates the feature weights

Init the coefficients $a_i = 0$

Until termination do

For each $\vec{x}_d \in D$, do

$o_d = \sum_{i=1}^{m} a_i t_i (\vec{x}_i, \vec{x}_d)$

if the classification is wrong,
that is $o_d t_d < 0$,
then

$a_d = a_d + \eta$

Updates the instance coefficients

With a mapping into higher dimensional space ….

This, again, is a heavy calculation.
But – if φ affords an efficient kernel K then we can replace it by a lower complexity kernel calculation.

Init the coefficients $a_i = 0$
For each $\vec{x}_d \in D$, compute $\varphi(\vec{x}_d)$
Until termination do
For each $\vec{x}_d \in D$, do

$$o_d = \sum_{i=1}^{m} a_i t_i \left(\varphi(\vec{x}_i), \varphi(\vec{x}_d)\right)$$

if the classification is wrong, that is $o_d t_d < 0$, then

$$a_d = a_d + \eta$$

The Kernel Perceptron

Init the coefficients $a_i = 0$

~~For each $\vec{x}_d \in D$, compute $\varphi(\vec{x}_d)$~~

Until termination do

For each $\vec{x}_d \in D$, do

$$o_d = \sum_{i=1}^{m} a_i t_i K(\vec{x}_i, \vec{x}_d)$$

if the classification is wrong, that is $o_d t_d < 0$, then

$$a_d = a_d + \eta$$

Converges to a linear separator in data mapped to higher dimensions, if such exists.
Does this at computational cost that is commensurate with the dimension of the original feature space

Examples

Polynomial kernels enable full rational varieties:

RBF kernels, representing infinite dimensional mapping:

$$K(\vec{x}, \vec{y}) = \exp\left(-\frac{\|\vec{x} - \vec{y}\|_2^2}{\sigma^2}\right)$$

Coming next: Large margins and SVMs

- Large margins and the geometry of the SVM optimization question
- How to solve constrained optimization problems. Lagrange Multipliers.
- How this yields SVM parameters (sketch)
- Slack variables and examples