An Introduction to Machine Learning

L3: Perceptron and Kernels

Alexander J. Smola

Statistical Machine Learning Program
Canberra, ACT 0200 Australia
Alex.Smola@nicta.com.au

Tata Institute, Pune, January 2007

Overview

L1: Machine learning and probability theory
Introduction to pattern recognition, classification, regression, novelty detection, probability theory, Bayes rule, inference

L2: Density estimation and Parzen windows
Nearest Neighbor, Kernels density estimation, Silverman's rule, Watson Nadaraya estimator, crossvalidation

L3: Perceptron and Kernels
Hebb's rule, perceptron algorithm, convergence, feature maps, kernels

L4: Support Vector estimation
Geometrical view, dual problem, convex optimization, kernels

L5: Support Vector estimation
Regression, Quantile regression, Novelty detection, ν-trick

L6: Structured Estimation
Sequence annotation, web page ranking, path planning, implementation and optimization

L3 Perceptron and Kernels

Hebb's rule

- positive feedback
- perceptron convergence rule

Hyperplanes

- Linear separability
- Inseparable sets

Features

- Explicit feature construction
- Implicit features via kernels

Kernels

- Examples
- Kernel perceptron

Biology and Learning

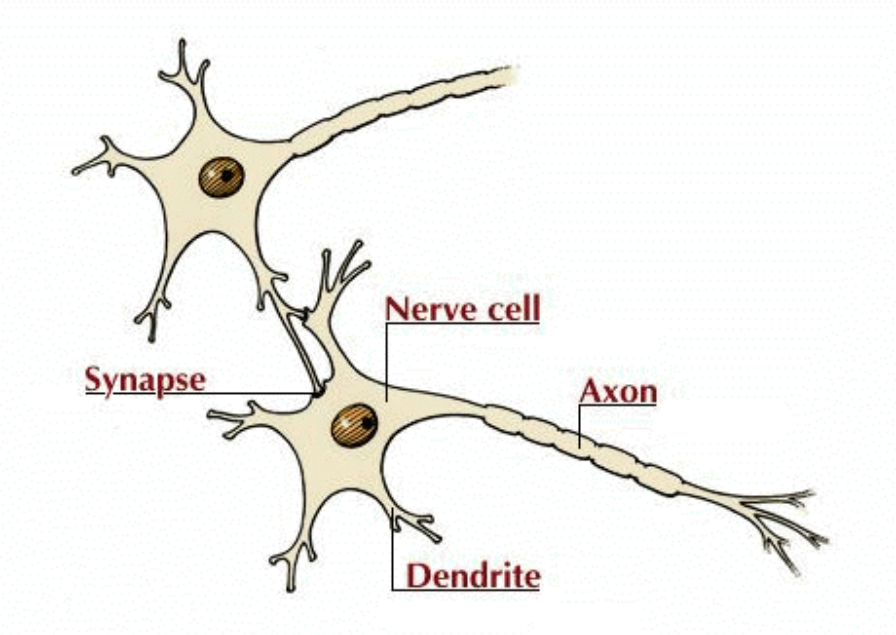
Basic Idea

- Good behavior should be rewarded, bad behavior punished (or not rewarded).
  This improves the fitness of the system.
- Example: hitting a tiger should be rewarded . . .
- Correlated events should be combined.
- Example: Pavlov's salivating dog.

Training Mechanisms

- Behavioral modification of individuals (learning):
  Successful behavior is rewarded (e.g. food).
- Hard-coded behavior in the genes (instinct):
  The wrongly coded animal dies.

Neurons



Soma

Cell body. Here the signals are combined ("CPU").

Dendrite

Combines the inputs from several other nerve cells ("input bus").

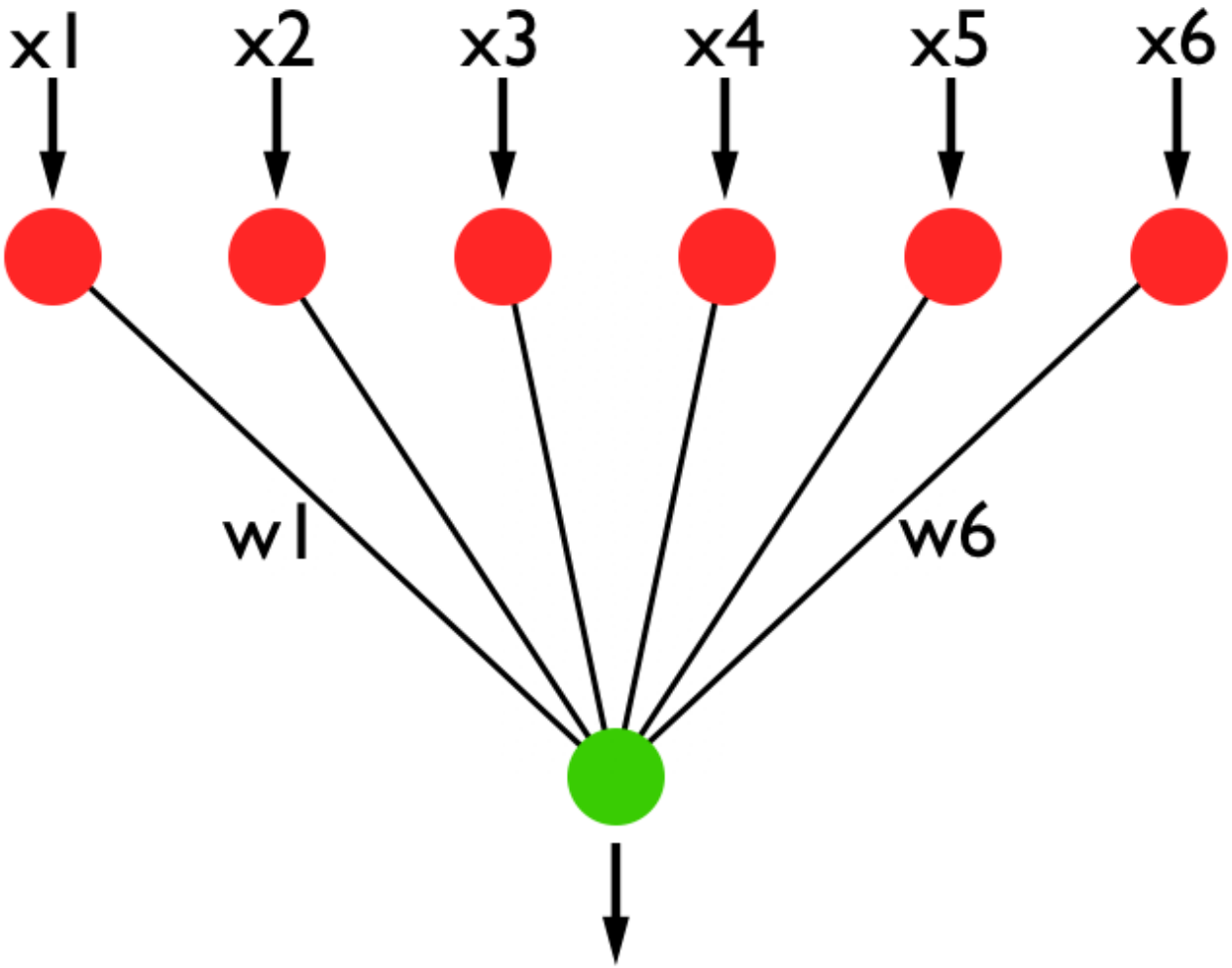
Synapse

Interface between two neurons ("connector").

Axon

This may be up to 1m long and will transport the activation signal to nerve cells at different locations ("output cable").

Perceptron



$$f(x) = w_1 x_1 + \ldots + w_6 x_6$$

Perceptrons

Weighted combination

- The output of the neuron is a linear combination of the inputs (from the other neurons via their axons) rescaled by the synaptic weights.
- Often the output does not directly correspond to the activation level but is a monotonic function thereof.

Decision Function

- At the end the results are combined into

$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right).$$

Separating Half Spaces

Linear Functions

An abstract model is to assume that

$$f(x) = \langle w, x \rangle + b$$

where $w, x \in \mathbb{R}^m$ and $b \in \mathbb{R}$.

Biological Interpretation

The weights w_i correspond to the synaptic weights (activating or inhibiting), the multiplication corresponds to the processing of inputs via the synapses, and the summation is the combination of signals in the cell body (soma).

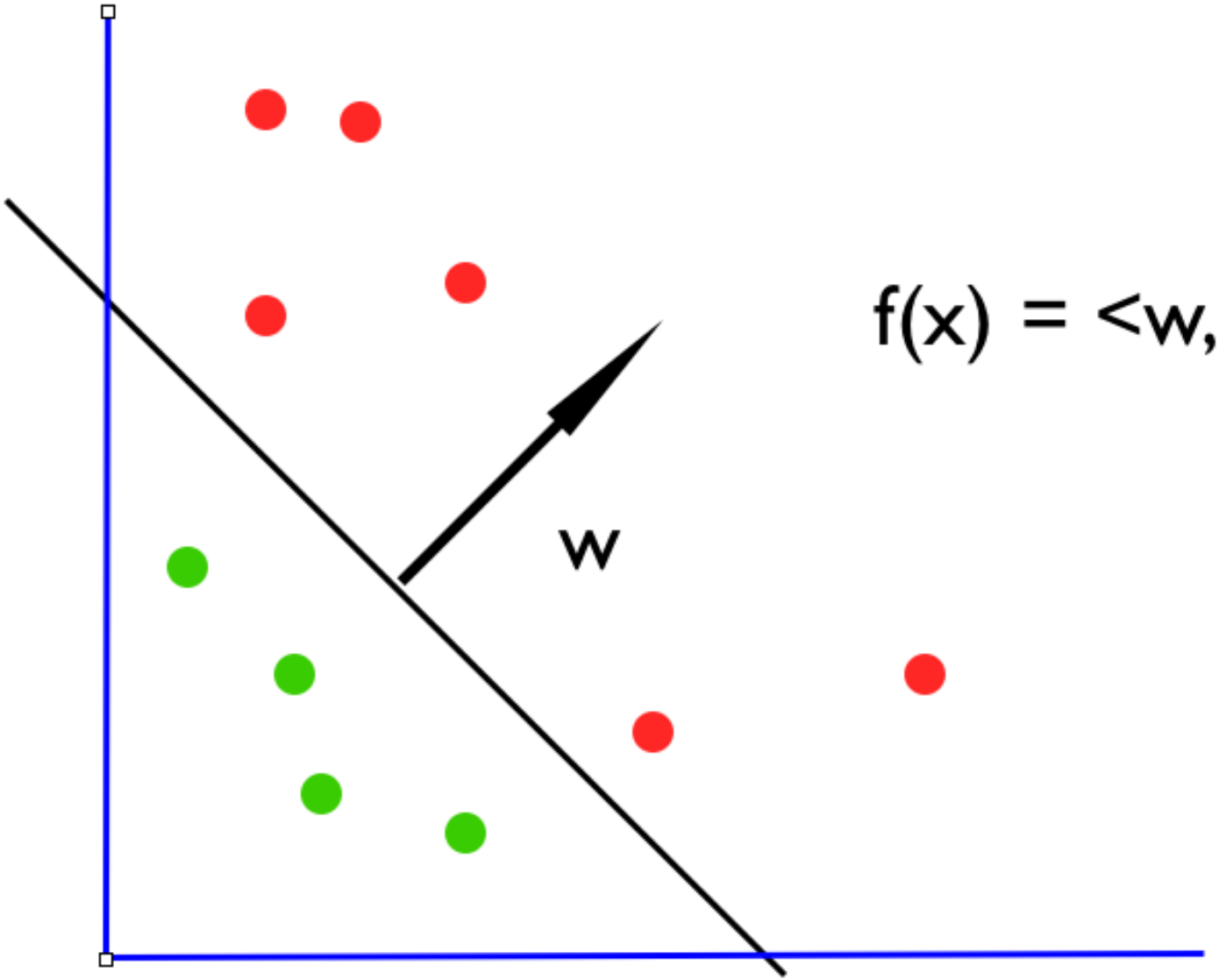
Applications

Spam filtering (e-mail), echo cancellation (old analog overseas cables)

Learning

Weights are “plastic” — adapted via the training data.

Linear Separation



$$f(x) = \langle w,x \rangle + b$$

Perceptron Algorithm

argument: $X := \{x_1, \ldots, x_m\} \subset \mathcal{X}$ (data)
$Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\}$ (labels)

function $(w, b) = \mathrm{Perceptron}(X, Y)$

initialize $w, b = 0$

repeat

Pick (x_i, y_i) from data

if $y_i(w \cdot x_i + b) \leq 0$ then

$w' = w + y_i x_i$

$b' = b + y_i$

until $y_i(w \cdot x_i + b) > 0$ for all i

end

Interpretation

Algorithm

- Nothing happens if we classify (x_i, y_i) correctly
- If we see incorrectly classified observation we update (w, b) by $y_i(x_i, 1)$.
- Positive reinforcement of observations.

Solution

- Weight vector is linear combination of observations x_i:

$$w \longleftarrow w + y_i x_i$$

- Classification can be written in terms of dot products:

$$w \cdot x + b = \sum_{j \in E} y_j x_j \cdot x + b$$

Theoretical Analysis

Incremental Algorithm

Already while the perceptron is learning, we can use it.

Convergence Theorem (Rosenblatt and Novikoff)

Suppose that there exists a $\rho > 0$, a weight vector w^* satisfying $\|w^*\| = 1$, and a threshold b^* such that

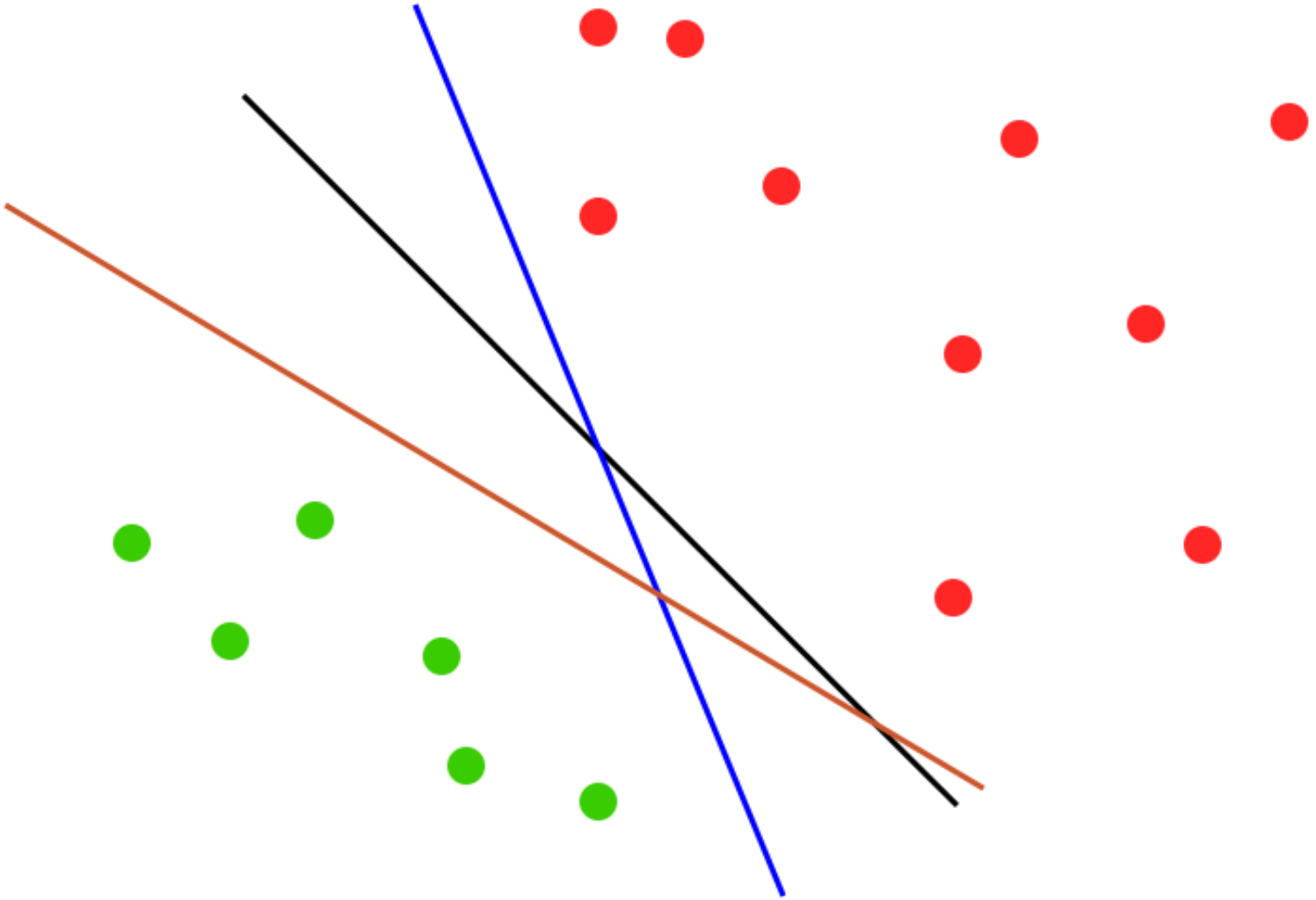
$$y_i \left(\langle w^*, x_i \rangle + b^* \right) \geq \rho \text{ for all } 1 \leq i \leq m.$$

Then the hypothesis maintained by the perceptron algorithm converges to a linear separator after no more than

$$\frac{(b^{*2} + 1)(R^2 + 1)}{\rho^2}$$

updates, where $R = \max_i \|x_i\|$.

Solutions of the Perceptron



Interpretation

Learning Algorithm

We perform an update only if we make a mistake.

Convergence Bound

- Bounds the maximum number of mistakes **in total**. We will make at most $(b^{*2} + 1)(R^1 + 1)/\rho^2$ mistakes in the case where a "correct" solution w^*, b^* exists.
- This also bounds the expected error (if we know ρ, R, and $|b^*|$).

Dimension Independent

Bound does not depend on the dimensionality of $\mathcal{X}$.

Sample Expansion

We obtain x as a **linear combination** of x_i.

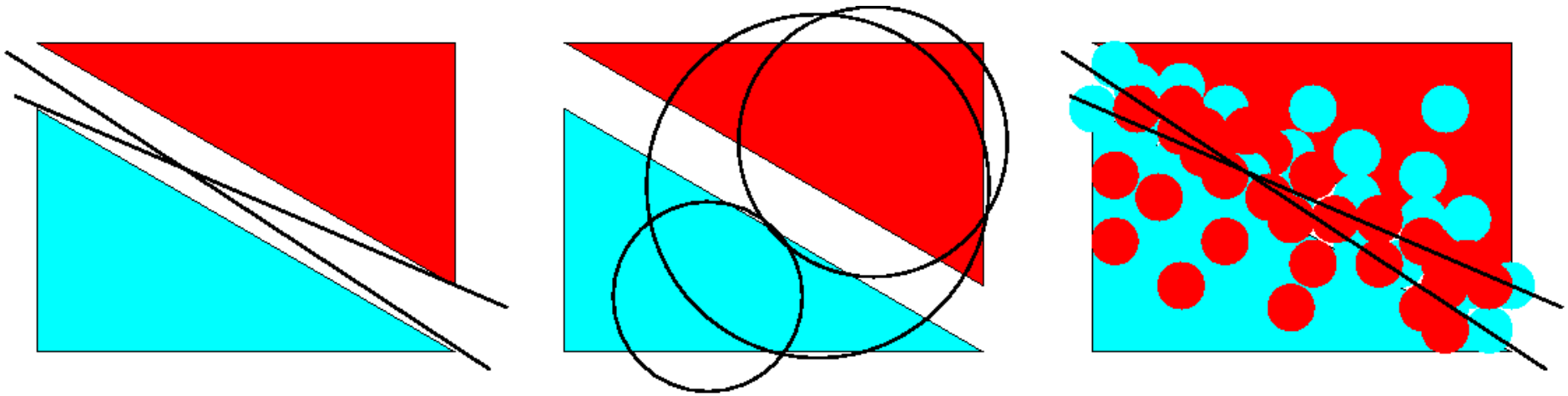
Realizable and Non-realizable Concepts

Realizable Concept

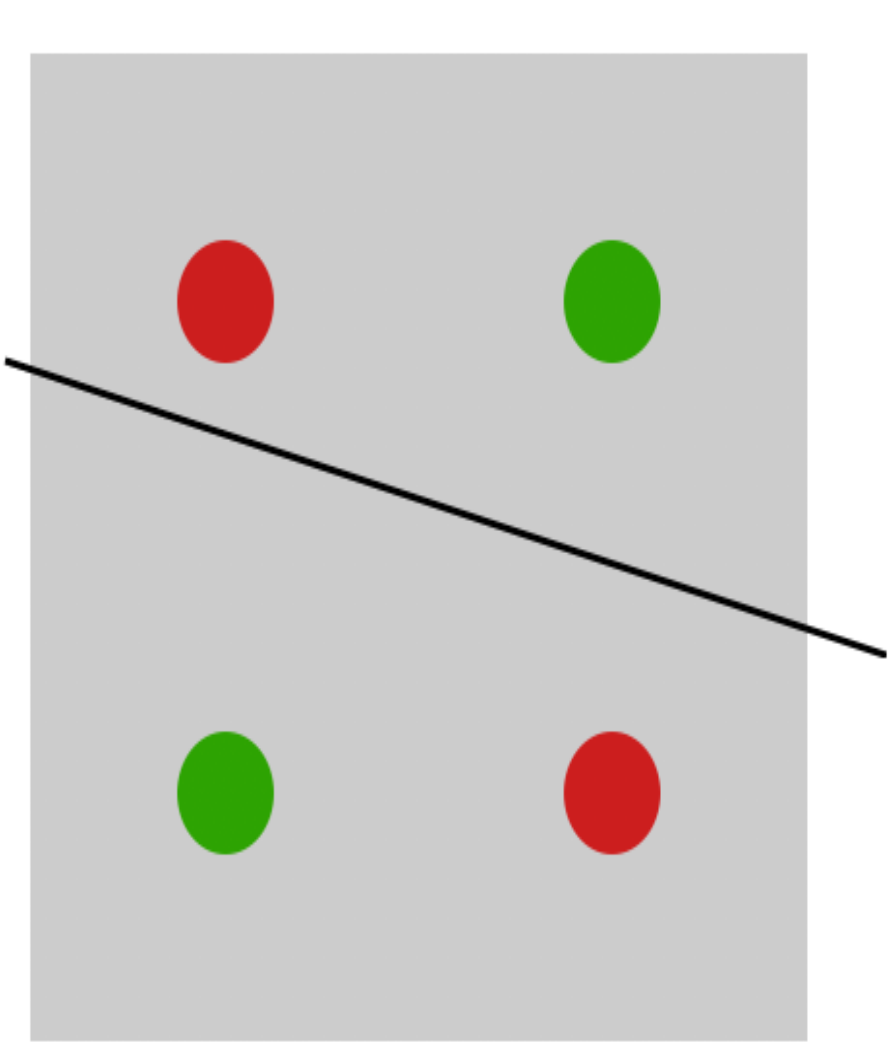
Here some w^*, b^* exists such that y is generated by $y = \operatorname{sgn}(\langle w^*, x \rangle + b)$. In general realizable means that the exact functional dependency is included in the class of admissible hypotheses.

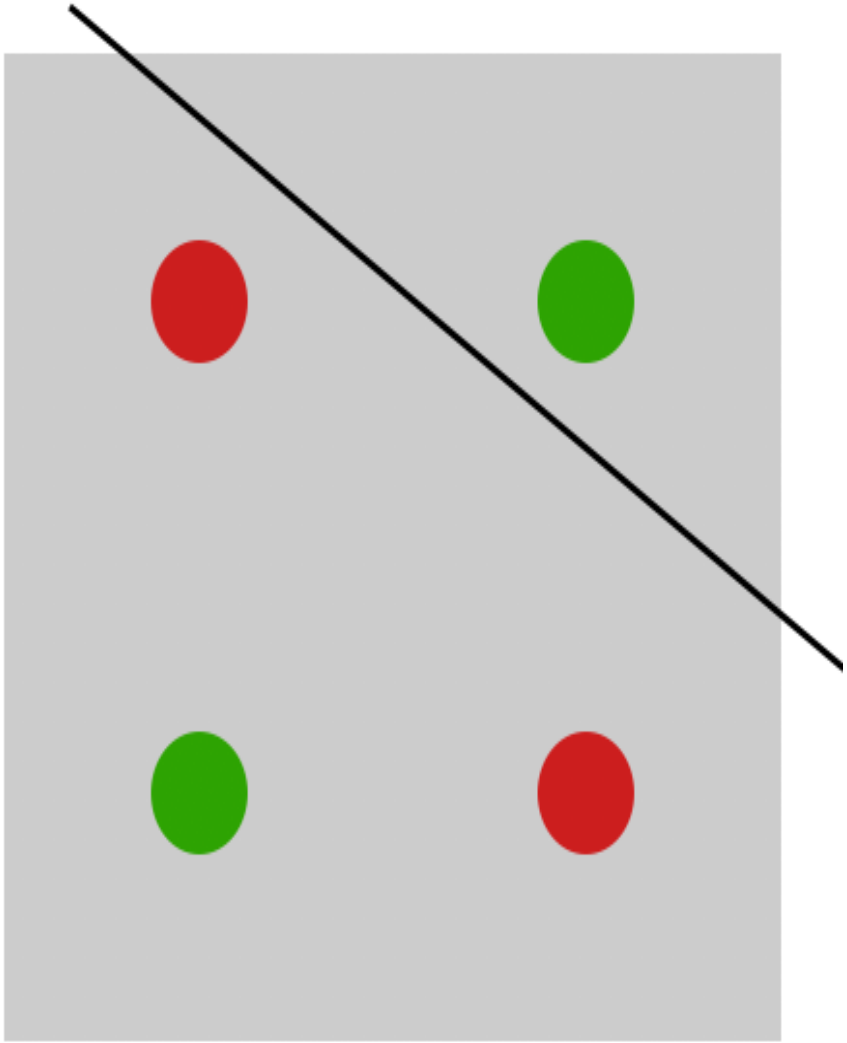
Unrealizable Concept

In this case, the exact concept does not exist or it is not included in the function class.



The XOR Problem





Mini Summary

Perceptron

- Separating halfspaces
- Perceptron algorithm
- Convergence theorem
- Only depends on margin, dimension independent

Pseudocode

```
for i in range(m):
    ytest = numpy.dot(w, x[:,i]) + b
    if ytest * y[i] <= 0:
        w += y[i] * x[:,i]
        b += y[i]
```

Nonlinearity via Preprocessing

Problem

Linear functions are often too simple to provide good estimators.

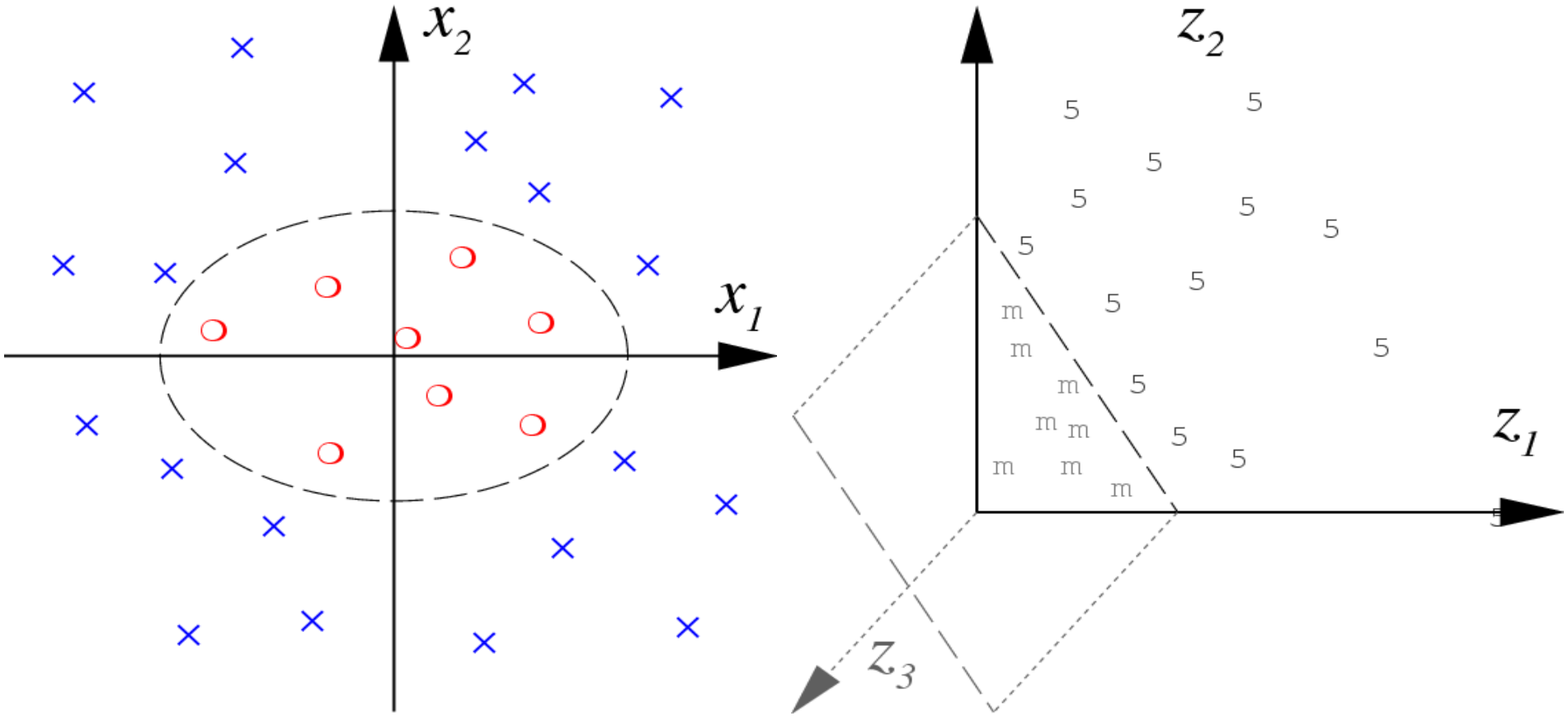
Idea

- Map to a higher dimensional feature space via $\Phi : x \rightarrow \Phi(x)$ and solve the problem there.
- Replace every $\langle x, x' \rangle$ by $\langle \Phi(x), \Phi(x') \rangle$ in the perceptron algorithm.

Consequence

- We have nonlinear classifiers.
- Solution lies in the choice of features $\Phi(x)$.

Nonlinearity via Preprocessing



Features

Quadratic features correspond to circles, hyperbolas and ellipsoids as separating surfaces.

Constructing Features

Idea

Construct features manually. E.g. for OCR we could use

	1	2	3	4	5	6	7	8	9	0
Loops	0	0	0	1	0	1	0	2	1	1
3 Joints	0	0	0	0	0	1	0	0	1	0
4 Joints	0	0	0	1	0	0	0	1	0	0
Angles	0	1	1	1	1	0	1	0	0	0
Ink	1	2	2	2	2	2	1	3	2	2

More Examples

Two Interlocking Spirals

If we transform the data (x_1, x_2) into a radial part ($r = \sqrt{x_1^2 + x_2^2}$) and an angular part ($x_1 = r\cos\phi$, $x_1 = r\sin\phi$), the problem becomes much easier to solve (we only have to distinguish different stripes).

Japanese Character Recognition

Break down the images into strokes and recognize it from the latter (there's a predefined order of them).

Medical Diagnosis

Include physician's comments, knowledge about unhealthy combinations, features in EEG, ...

Suitable Rescaling

If we observe, say the weight and the height of a person, rescale to zero mean and unit variance.

Perceptron on Features

argument: $X := \{x_1, \ldots, x_m\} \subset \mathcal{X}$ (data)
$Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\}$ (labels)

function $(w, b) = \mathrm{Perceptron}(X, Y, \eta)$

initialize $w, b = 0$

repeat

Pick (x_i, y_i) from data

if $y_i(w \cdot \Phi(x_i) + b) \leq 0$ then

$w' = w + y_i \Phi(x_i)$

$b' = b + y_i$

until $y_i(w \cdot \Phi(x_i) + b) > 0$ for all i

end

Important detail

$w = \sum_j y_j \Phi(x_j)$ and hence $f(x) = \sum_j y_j (\Phi(x_j) \cdot \Phi(x)) + b$

Problems with Constructing Features

Problems

- Need to be an expert in the domain (e.g. Chinese characters).
- Features may not be robust (e.g. postman drops letter in dirt).
- Can be expensive to compute.

Solution

- Use shotgun approach.
- Compute many features and hope a good one is among them.
- Do this efficiently.

Polynomial Features

Quadratic Features in $\mathbb{R}^2$

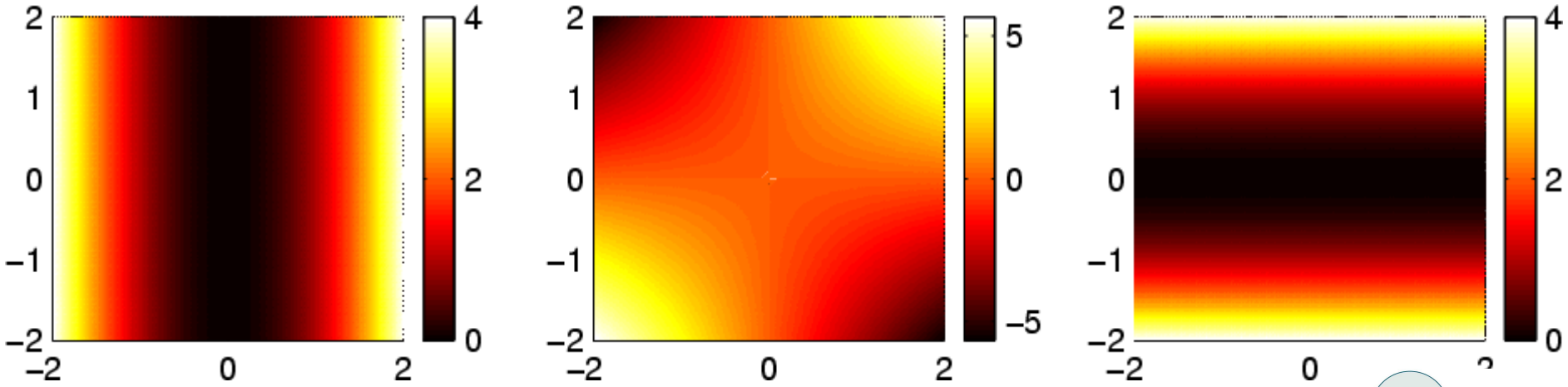
$$\Phi(x) := \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

Dot Product

$$\begin{aligned}\langle \Phi(x), \Phi(x') \rangle &= \left\langle \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right), \left({x'_1}^2, \sqrt{2}x'_1x'_2, {x'_2}^2\right) \right\rangle \\ &= \langle x, x' \rangle^2.\end{aligned}$$

Insight

Trick works for any polynomials of order d via $\langle x, x' \rangle^d$.



Kernels

Problem

- Extracting features can sometimes be very costly.
- Example: second order features in 1000 dimensions. This leads to 5005 numbers. For higher order polynomial features much worse.

Solution

Don't compute the features, try to compute dot products implicitly. For some features this works ...

Definition

A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function in its arguments for which the following property holds

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \text{ for some feature map } \Phi.$$

If $k(x, x')$ is much cheaper to compute than $\Phi(x)$ ...

Polynomial Kernels in $\mathbb{R}^n$

Idea

- We want to extend $k(x, x') = \langle x, x' \rangle^2$ to

$$k(x, x') = (\langle x, x' \rangle + c)^d \text{ where } c > 0 \text{ and } d \in \mathbb{N}.$$

- Prove that such a kernel corresponds to a dot product.

Proof strategy

Simple and straightforward: compute the explicit sum given by the kernel, i.e.

$$k(x, x') = (\langle x, x' \rangle + c)^d = \sum_{i=0}^{m} \binom{d}{i} (\langle x, x' \rangle)^i c^{d-i}$$

Individual terms $(\langle x, x' \rangle)^i$ are dot products for some $\Phi_i(x)$.

Kernel Perceptron

argument: $X := \{x_1, \ldots, x_m\} \subset \mathcal{X}$ (data)
$Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\}$ (labels)

function $f = \text{Perceptron}(X, Y, \eta)$

initialize $f = 0$

repeat

Pick (x_i, y_i) from data

if $y_i f(x_i) \leq 0$ then

$f(\cdot) \leftarrow f(\cdot) + y_i k(x_i, \cdot) + y_i$

until $y_i f(x_i) > 0$ for all i

end

Important detail

$w = \sum_j y_j \Phi(x_j)$ and hence $f(x) = \sum_j y_j k(x_j, x) + b$.

Are all $k(x, x')$ good Kernels?

Computability

We have to be able to compute $k(x, x')$ efficiently (much cheaper than dot products themselves).

"Nice and Useful" Functions

The features themselves have to be useful for the learning problem at hand. Quite often this means smooth functions.

Symmetry

Obviously $k(x, x') = k(x', x)$ due to the symmetry of the dot product $\langle \Phi(x), \Phi(x') \rangle = \langle \Phi(x'), \Phi(x) \rangle$.

Dot Product in Feature Space

Is there always a Φ such that k really is a dot product?

Mercer's Theorem

The Theorem

For any symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is square integrable in $\mathcal{X} \times \mathcal{X}$ and which satisfies

$$\int_{\mathcal{X}\times\mathcal{X}} k(x, x')f(x)f(x')dxdx' \geq 0 \text{ for all } f \in L_2(\mathcal{X})$$

there exist $\phi_i : \mathcal{X} \to \mathbb{R}$ and numbers $\lambda_i \geq 0$ where

$$k(x, x') = \sum_i \lambda_i \phi_i(x)\phi_i(x') \text{ for all } x, x' \in \mathcal{X}.$$

Interpretation

Double integral is continuous version of vector-matrix-vector multiplication. For positive semidefinite matrices

$$\sum_i \sum_j k(x_i, x_j)\alpha_i\alpha_j \geq 0$$

Properties of the Kernel

Distance in Feature Space

Distance between points in feature space via

$$\begin{aligned} d(x,x')^2 :=& \|\Phi(x) - \Phi(x')\|^2 \\ =& \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), \Phi(x') \rangle + \langle \Phi(x'), \Phi(x') \rangle \\ =& k(x,x) - 2k(x,x') + k(x',x') \end{aligned}$$

Kernel Matrix

To compare observations we compute dot products, so we study the matrix K given by

$$K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_i, x_j)$$

where x_i are the training patterns.

Similarity Measure

The entries K_{ij} tell us the overlap between $\Phi(x_i)$ and $\Phi(x_j)$, so $k(x_i, x_j)$ is a similarity measure.

Properties of the Kernel Matrix

K is Positive Semidefinite

Claim: $\alpha^\top K \alpha \geq 0$ for all $\alpha \in \mathbb{R}^m$ and all kernel matrices $K \in \mathbb{R}^{m \times m}$. Proof:

$$\sum_{i,j}^{m} \alpha_i \alpha_j K_{ij} = \sum_{i,j}^{m} \alpha_i \alpha_j \langle \Phi(x_i), \Phi(x_j) \rangle$$

$$= \left\langle \sum_{i}^{m} \alpha_i \Phi(x_i), \sum_{j}^{m} \alpha_j \Phi(x_j) \right\rangle = \left\| \sum_{i=1}^{m} \alpha_i \Phi(x_i) \right\|^2$$

Kernel Expansion

If w is given by a linear combination of $\Phi(x_i)$ we get

$$\langle w, \Phi(x) \rangle = \left\langle \sum_{i=1}^{m} \alpha_i \Phi(x_i), \Phi(x) \right\rangle = \sum_{i=1}^{m} \alpha_i k(x_i, x).$$

A Counterexample

A Candidate for a Kernel

$$k(x, x') = \begin{cases} 1 & \text{if } \|x - x'\| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

This is symmetric and gives us some information about the proximity of points, yet it is not a proper kernel ...

Kernel Matrix

We use three points, $x_1 = 1, x_2 = 2, x_3 = 3$ and compute the resulting "kernelmatrix" K. This yields

$$K = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$ and eigenvalues $(\sqrt{2}-1)^{-1}, 1$ and $(1-\sqrt{2})$.

as eigensystem. Hence k is not a kernel.

Some Good Kernels

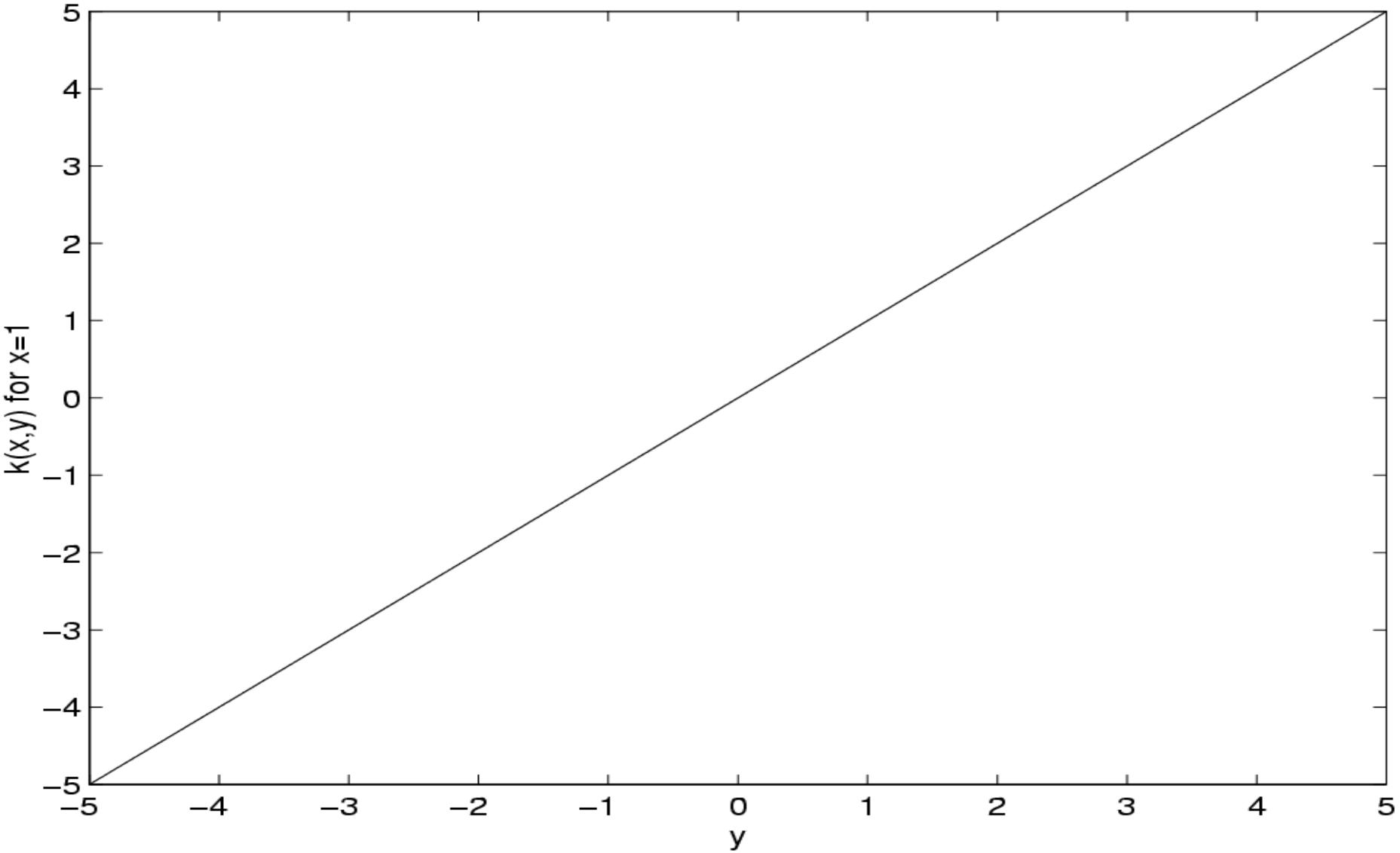
Examples of kernels $k(x, x')$

Linear	$\langle x, x' \rangle$
Laplacian RBF	$\exp\left(-\lambda \|x - x'\|\right)$
Gaussian RBF	$\exp\left(-\lambda \|x - x'\|^2\right)$
Polynomial	$\left(\langle x, x' \rangle + c\rangle\right)^d, c \geq 0,\ d \in \mathbb{N}$
B-Spline	$B_{2n+1}(x - x')$
Cond. Expectation	$\mathbf{E}_c[p(x\|c)p(x'\|c)]$

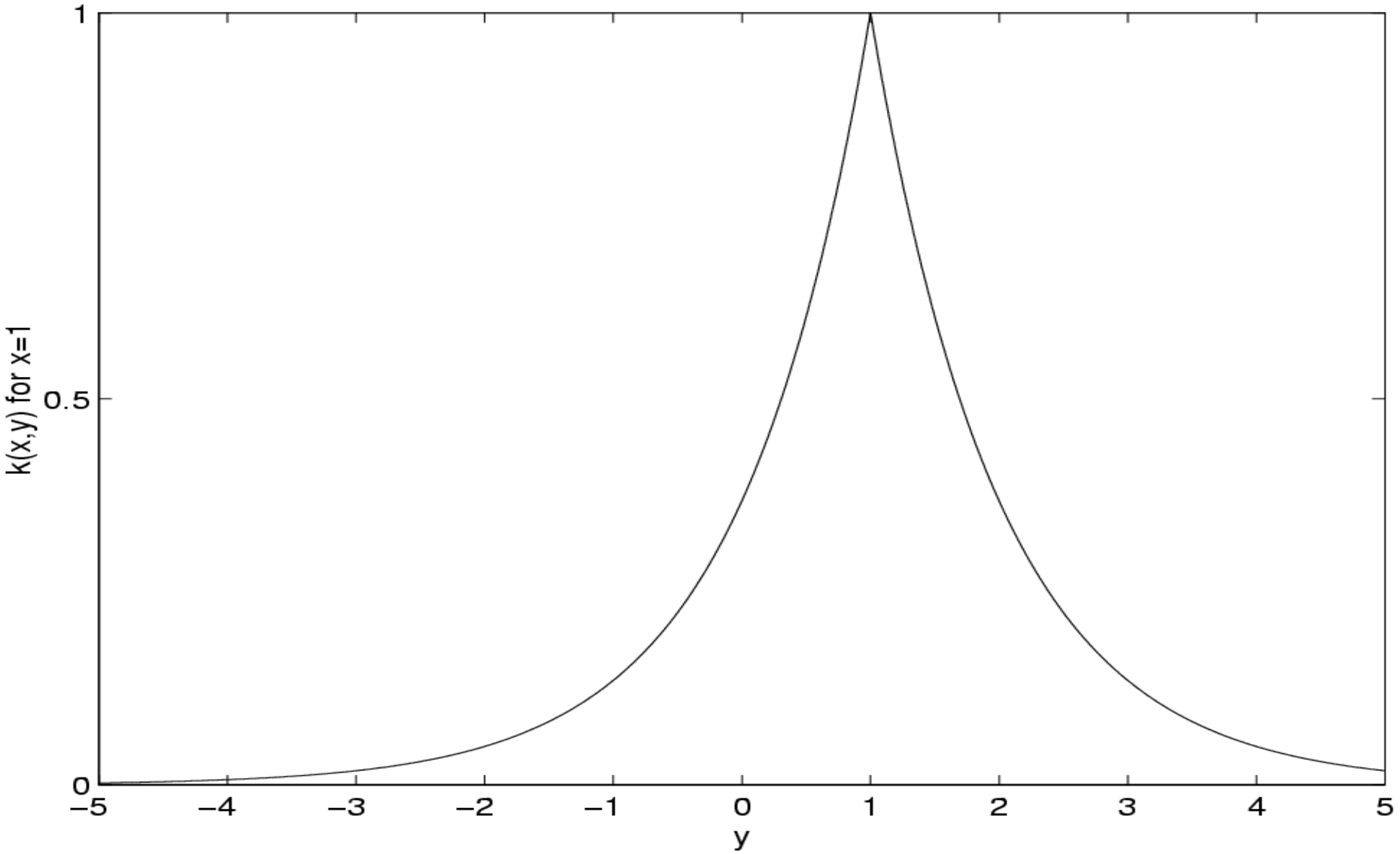
Simple trick for checking Mercer's condition

Compute the Fourier transform of the kernel and check that it is nonnegative.

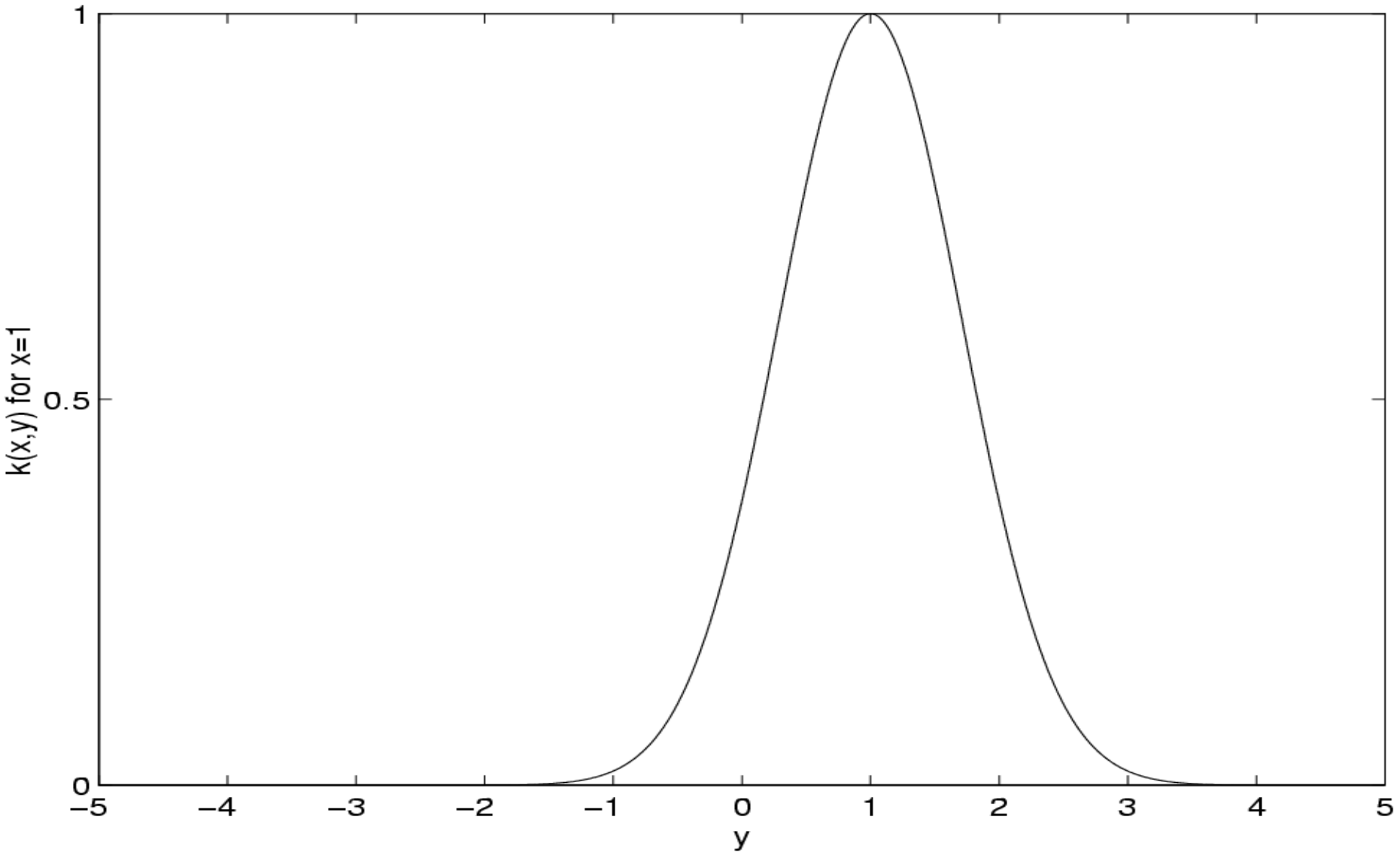
Linear Kernel



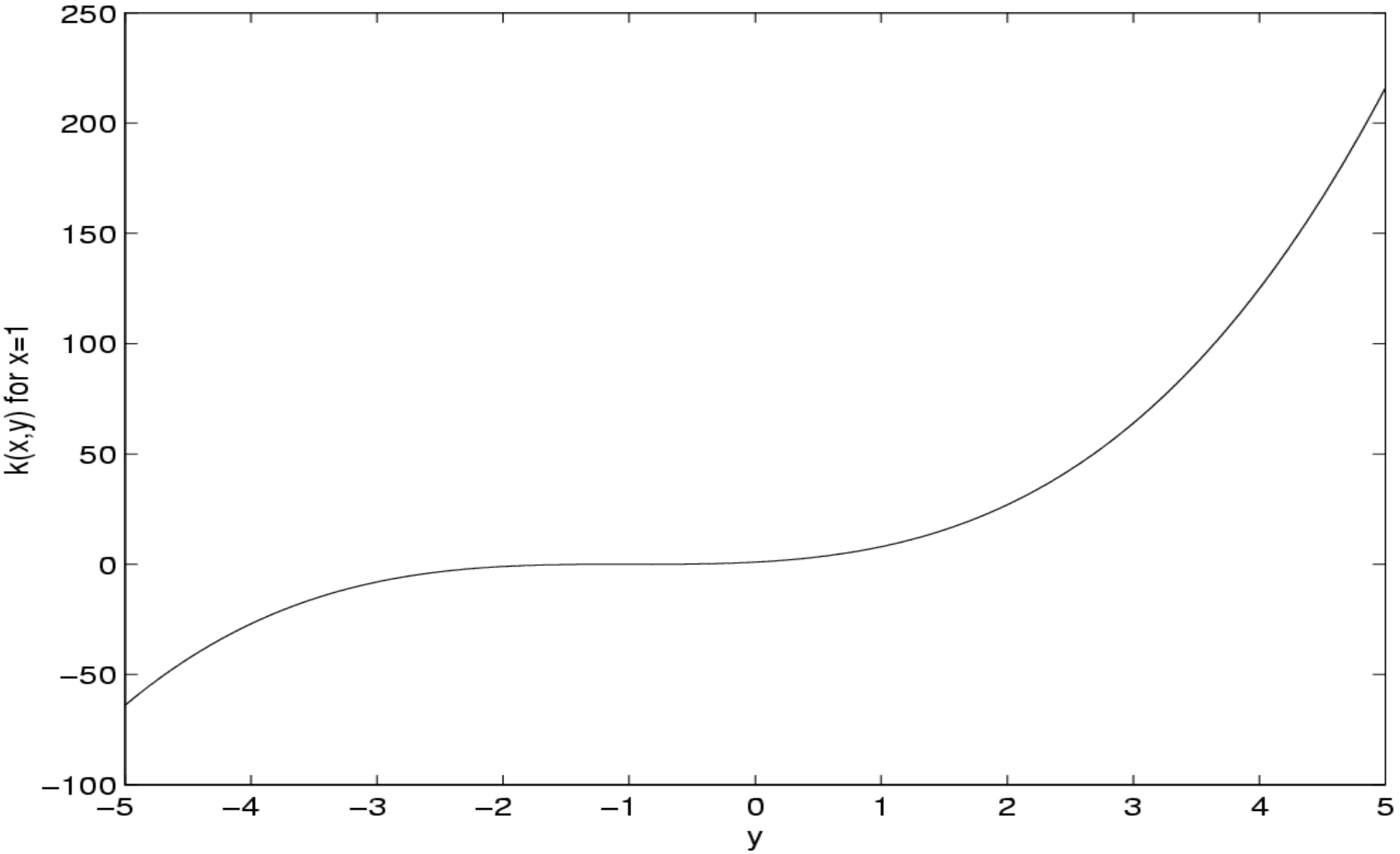
Laplacian Kernel



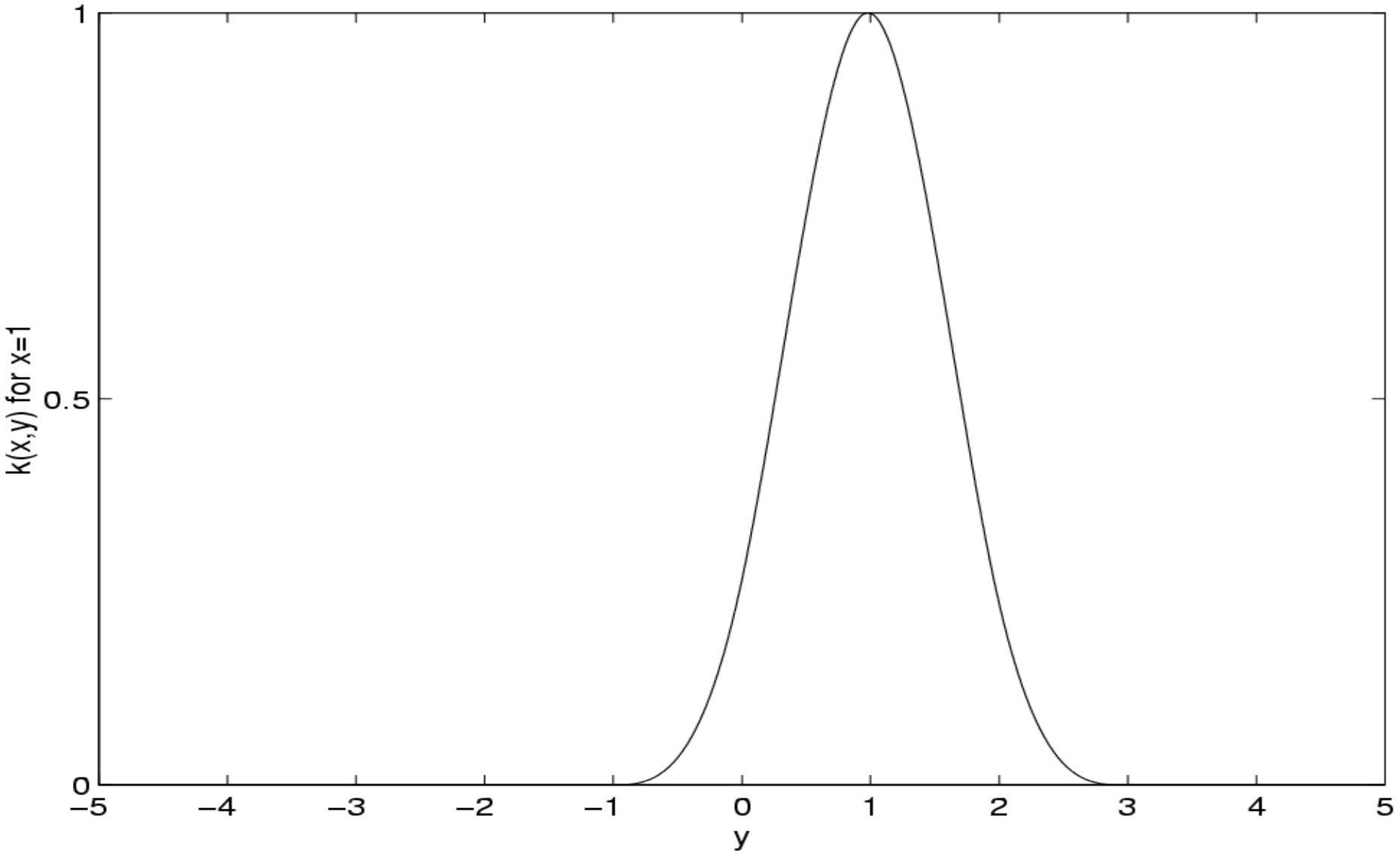
Gaussian Kernel



Polynomial (Order 3)



B_3-Spline Kernel



Mini Summary

Features

- Prior knowledge, expert knowledge
- Shotgun approach (polynomial features)
- Kernel trick $k(x, x') = \langle \phi(x), \phi(x') \rangle$
- Mercer's theorem

Applications

- Kernel Perceptron
- Nonlinear algorithm automatically by query-replace

Examples of Kernels

- Gaussian RBF
- Polynomial kernels

Summary

Hebb's rule

- positive feedback
- perceptron convergence rule, kernel perceptron

Features

- Explicit feature construction
- Implicit features via kernels

Kernels

- Examples
- Mercer's theorem