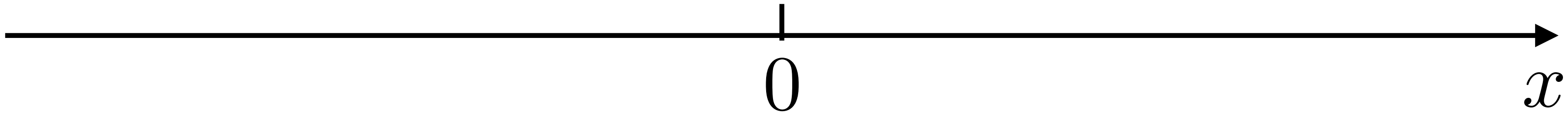
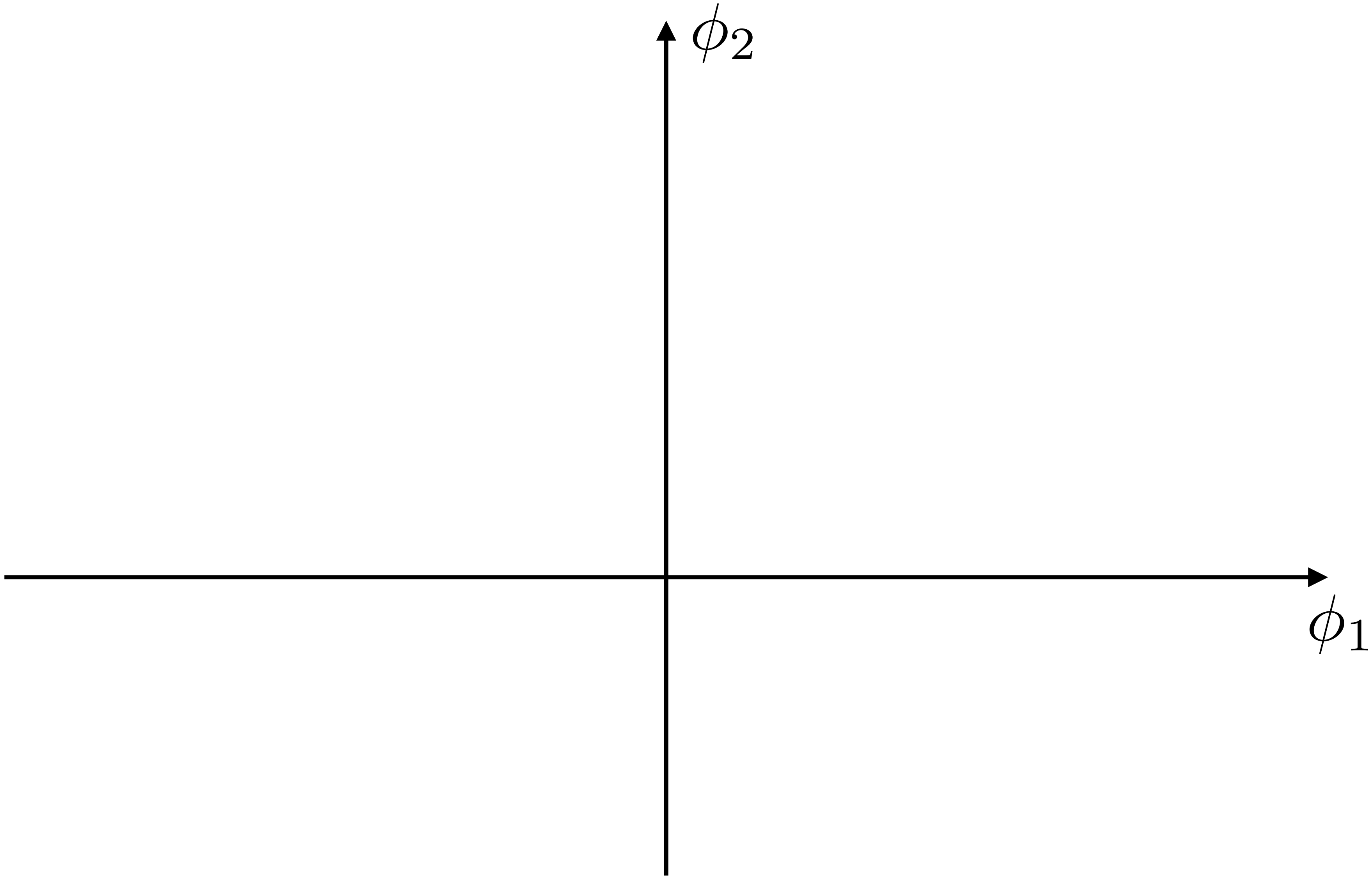
Outline

- Non-linear classification and regression
- Feature maps, their inner products
- Kernel functions induced from feature maps
- Kernel methods, kernel perceptron
- Other non-linear classifiers (e.g., Random Forest)

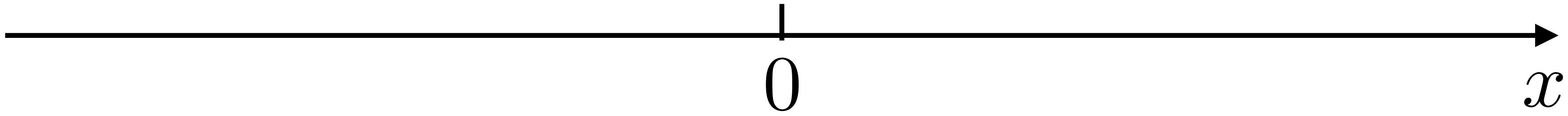
Linear classifiers on the real line



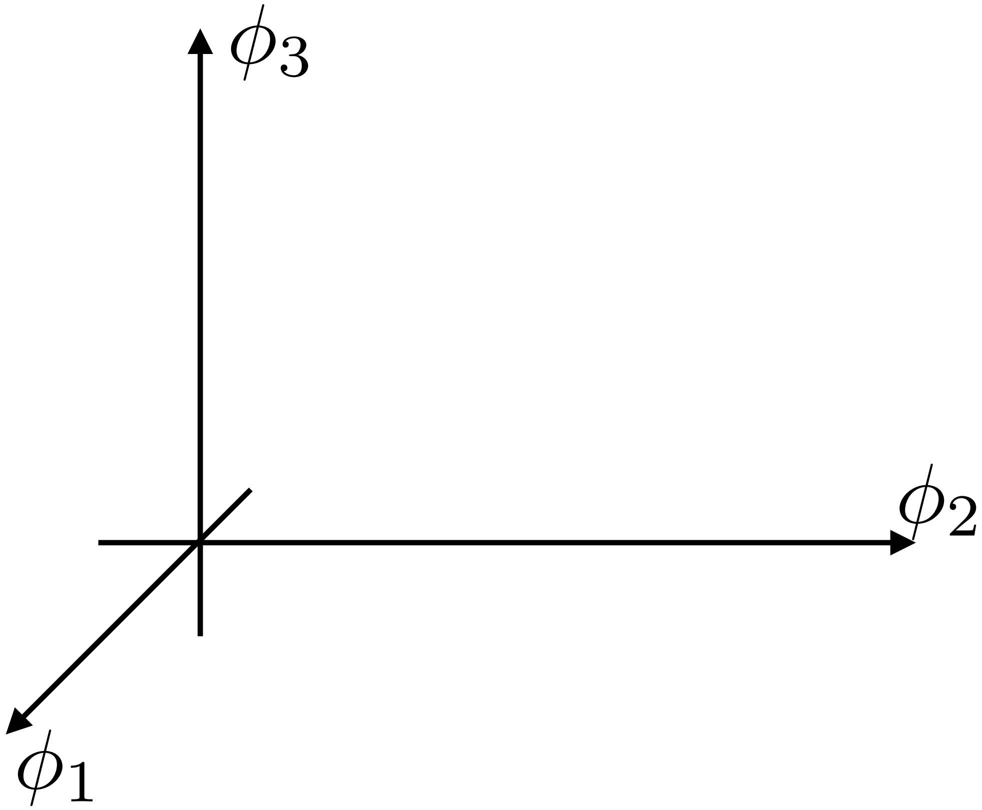
In feature space

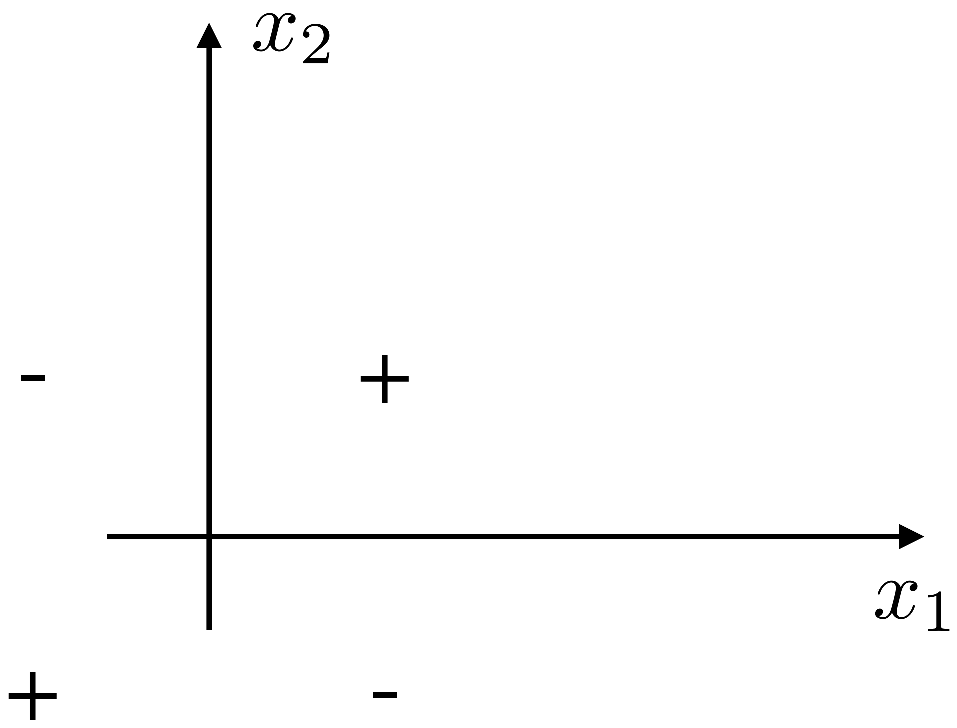


Back to the real line



2-dim example





Polynomial features

- We can add more polynomial terms

- Means lots of features in higher dimensions

Non-lin. classification & regression

- Non-linear classification

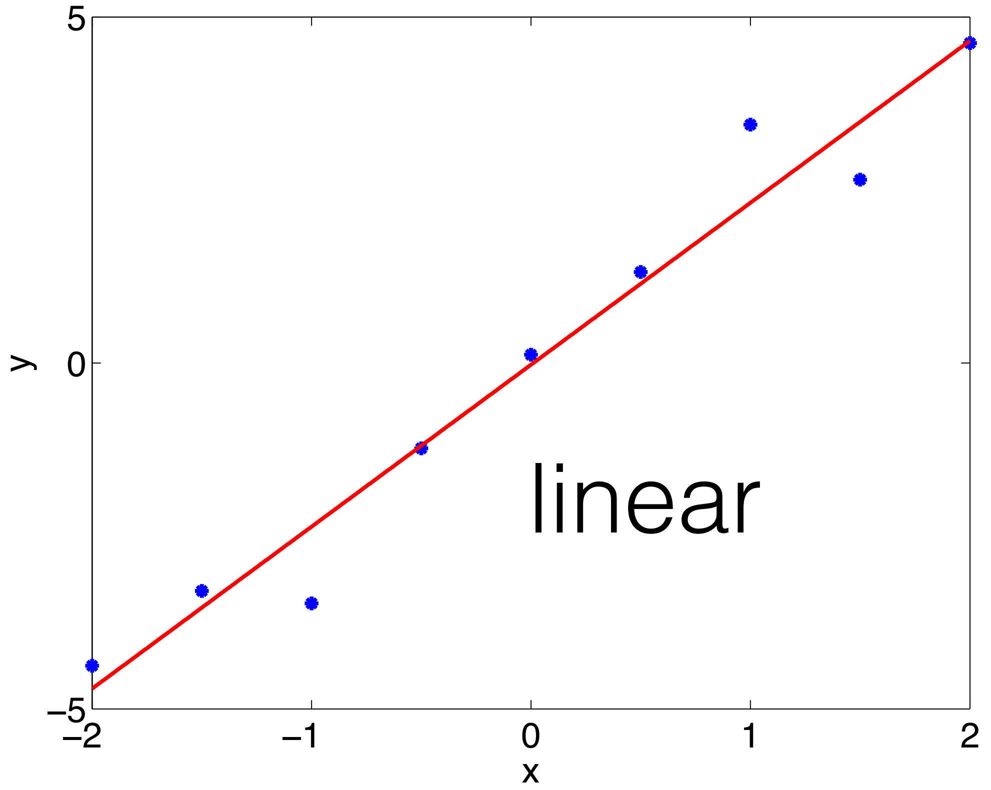
$$h(x; \theta, \theta_0) = \text{sign}\big(\theta \cdot \phi(x) + \theta_0\big)$$

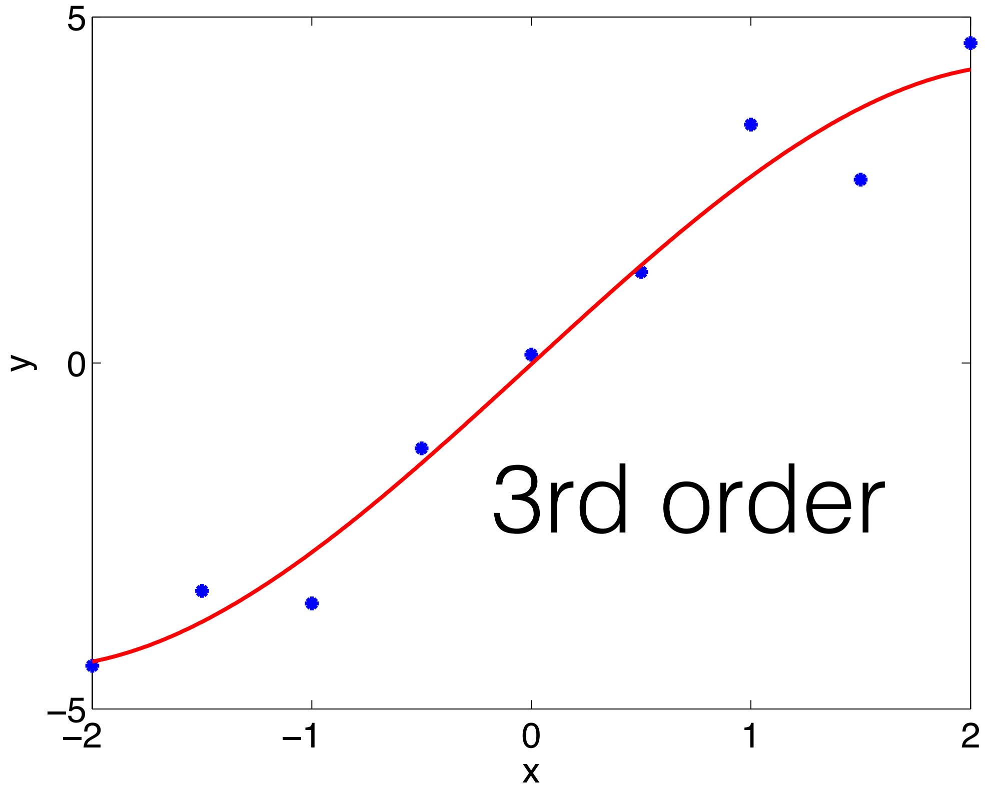
- Non-linear regression

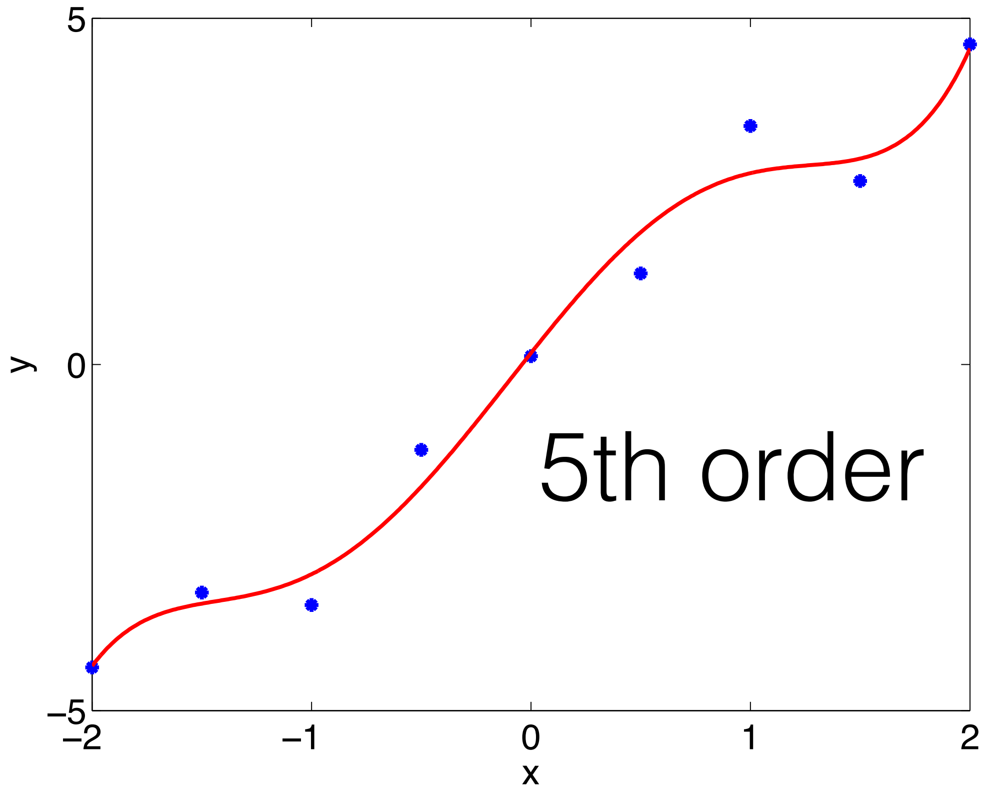
$$f(x; \theta, \theta_0) = \theta \cdot \phi(x) + \theta_0$$

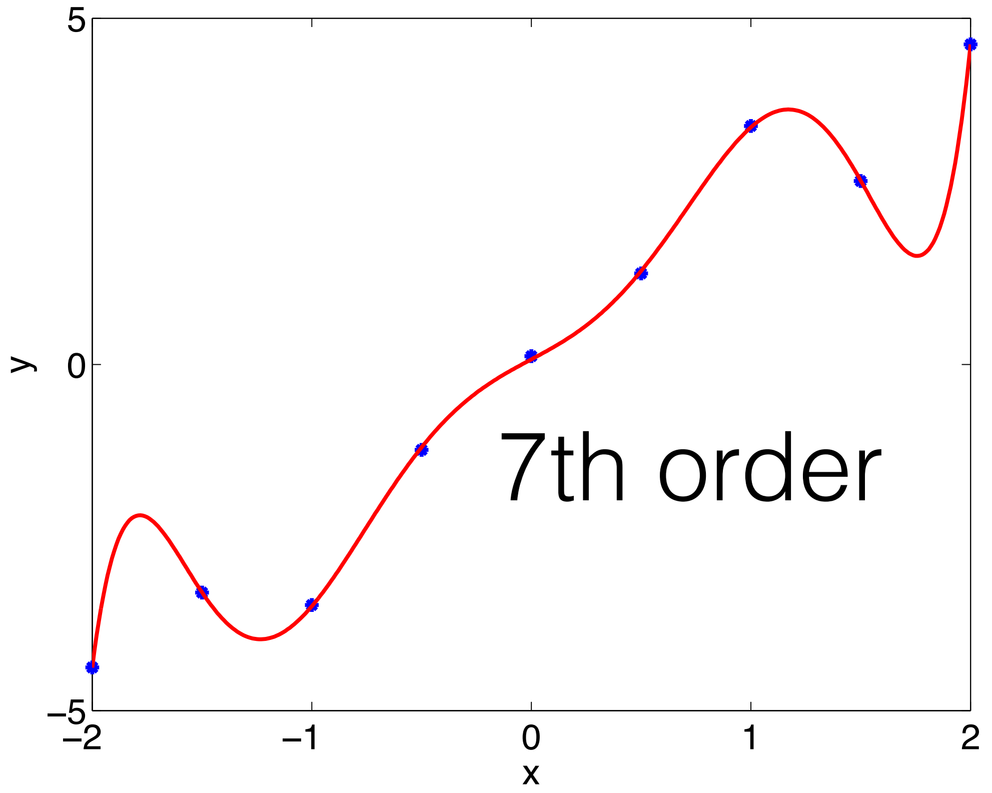
e.g., $\phi(x) = [x, x^2]^T$

Non-linear regression









Why not feature vectors?

- By mapping input examples explicitly into feature vectors, and performing linear classification or regression on top of such feature vectors, we get a lot of expressive power
- But the downside is that these vectors can be quite high dimensional

Inner products, kernels

- Computing the inner product between two feature vectors **can be** cheap even if the vectors are very high dimensional

$$\phi(x) = [x_1, x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2]^T$$

$$\phi(x') = [x'_1, x'_2, {x'}^2_1, \sqrt{2}x'_1x'_2, {x'}^2_2]^T$$

Kernels vs features

- For **some** feature maps, we can evaluate the inner products very efficiently, e.g.,
- In those cases, it is advantageous to express the linear classifiers (regression methods) in terms of kernels rather than explicitly constructing feature vectors

Recall perceptron

```
$\theta = 0$
run through $i = 1, \ldots, n$
    if $y^{(i)}\,\theta \cdot \phi(x^{(i)}) \leq 0$
        $\theta \leftarrow \theta + y^{(i)}\phi(x^{(i)})$
```

Recall perceptron

```
$\theta = 0$
run through $i = 1, \ldots, n$
        if $y^{(i)}\,\theta \cdot \phi(x^{(i)}) \leq 0$
            $\theta \leftarrow \theta + y^{(i)}\phi(x^{(i)})$
```

Feature engineering, kernels

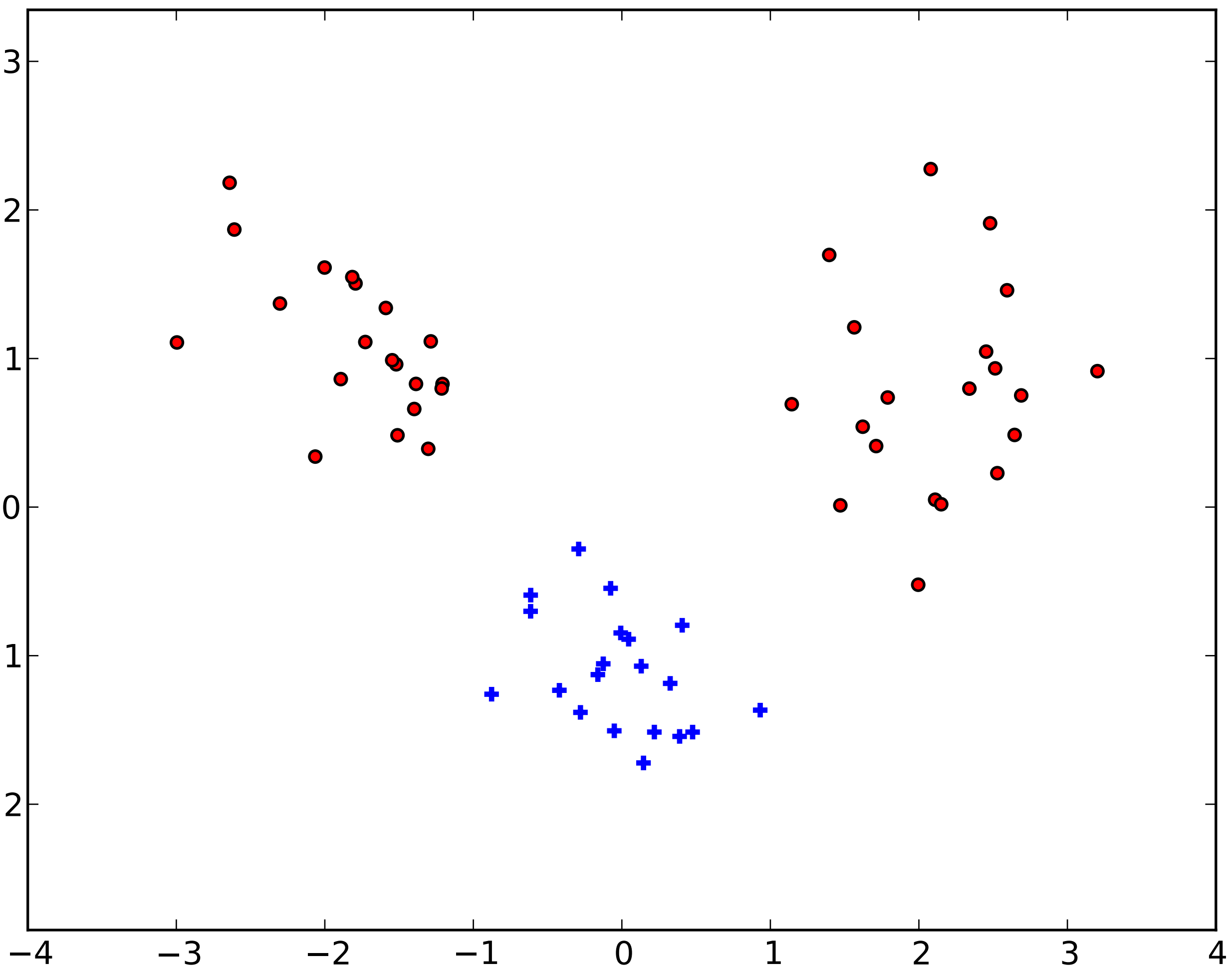
- Composition rules:

1. $K(x, x') = 1$ is a kernel function.
2. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $K(x, x')$ is a kernel. Then so is $\tilde{K}(x, x') = f(x)K(x, x')f(x')$
3. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel
4. If $K_1(x, x')$ and $K_2(x, x')$ are kernels, then $K(x, x') = K_1(x, x')K_2(x, x')$ is a kernel

Radial basis kernel

- The feature vectors can be infinite dimensional… this means that they have unlimited expressive power

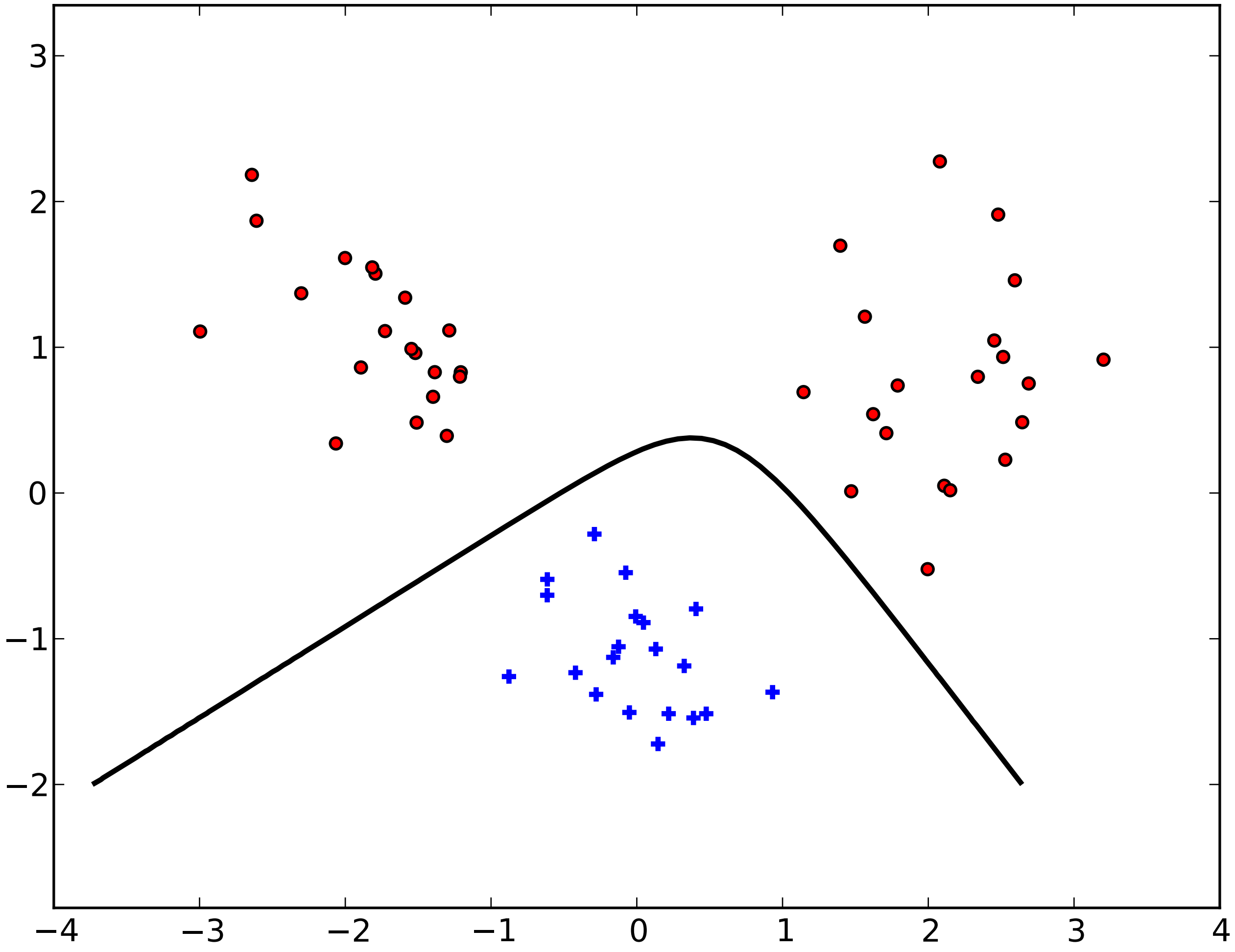
$$K(x, x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$



Radial basis kernel

- The feature vectors can be infinite dimensional… this means that they have unlimited expressive power

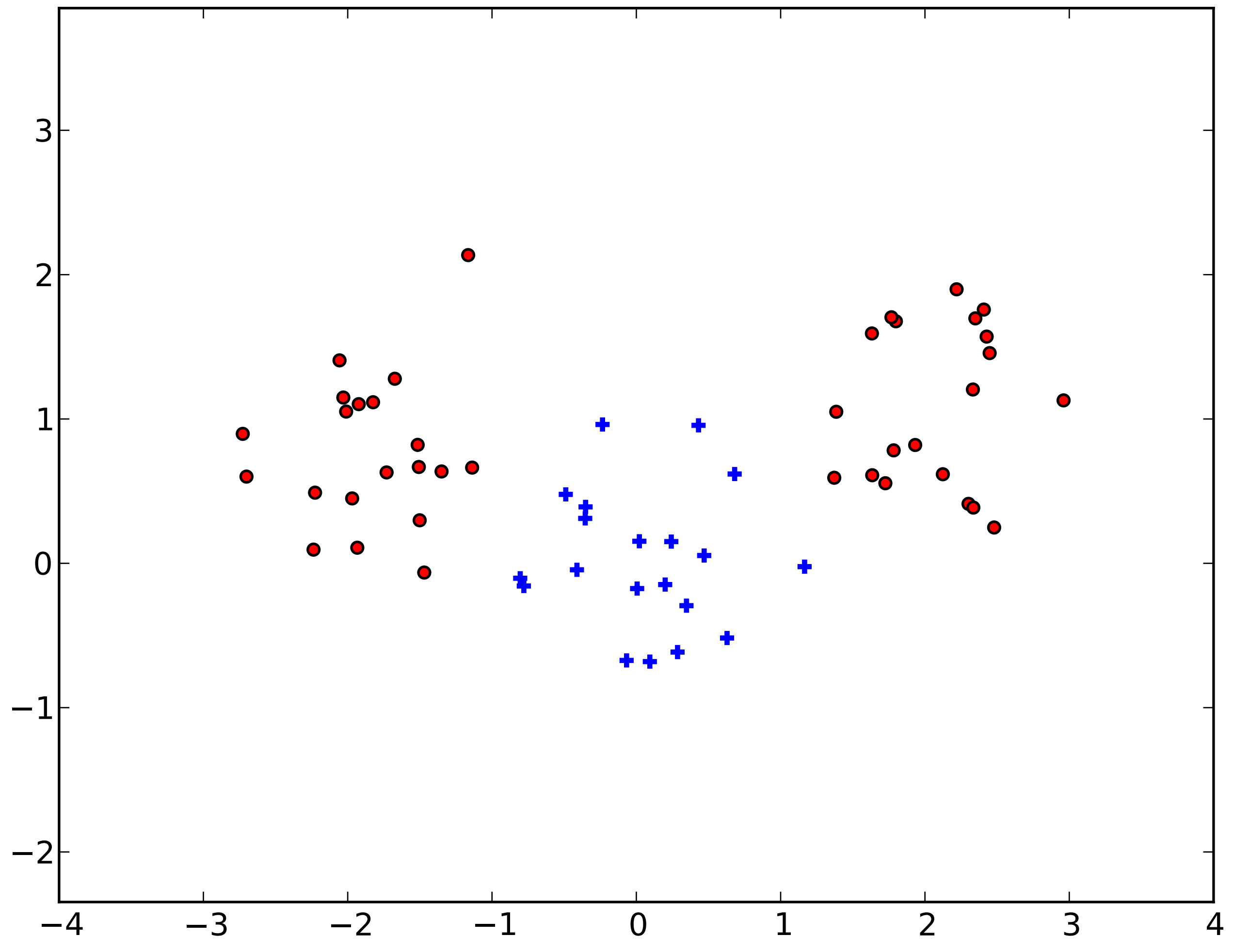
$$K(x, x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$



Radial basis kernel

- The feature vectors can be infinite dimensional… this means that they have unlimited expressive power

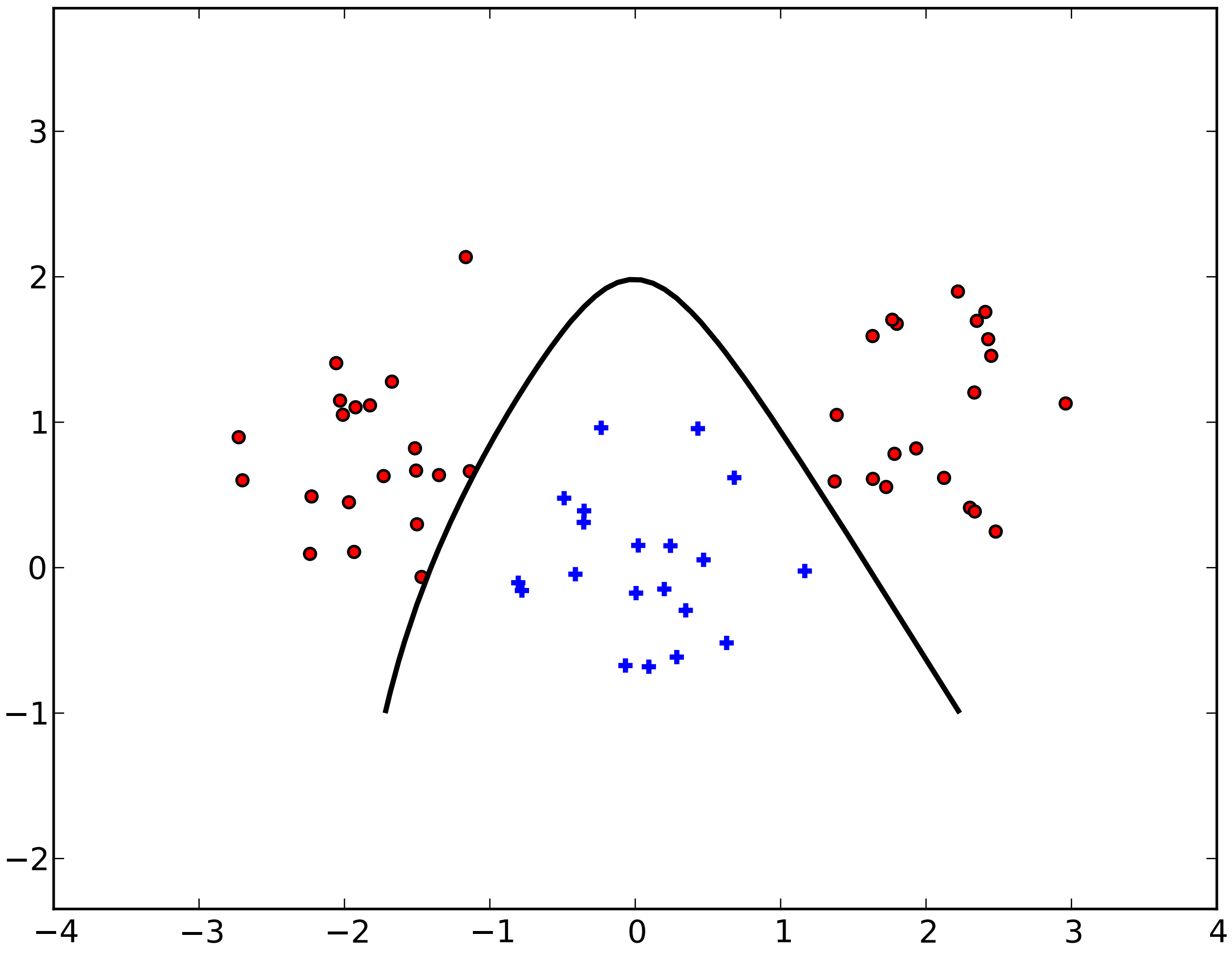
$$K(x, x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$



Radial basis kernel

- The feature vectors can be infinite dimensional… this means that they have unlimited expressive power

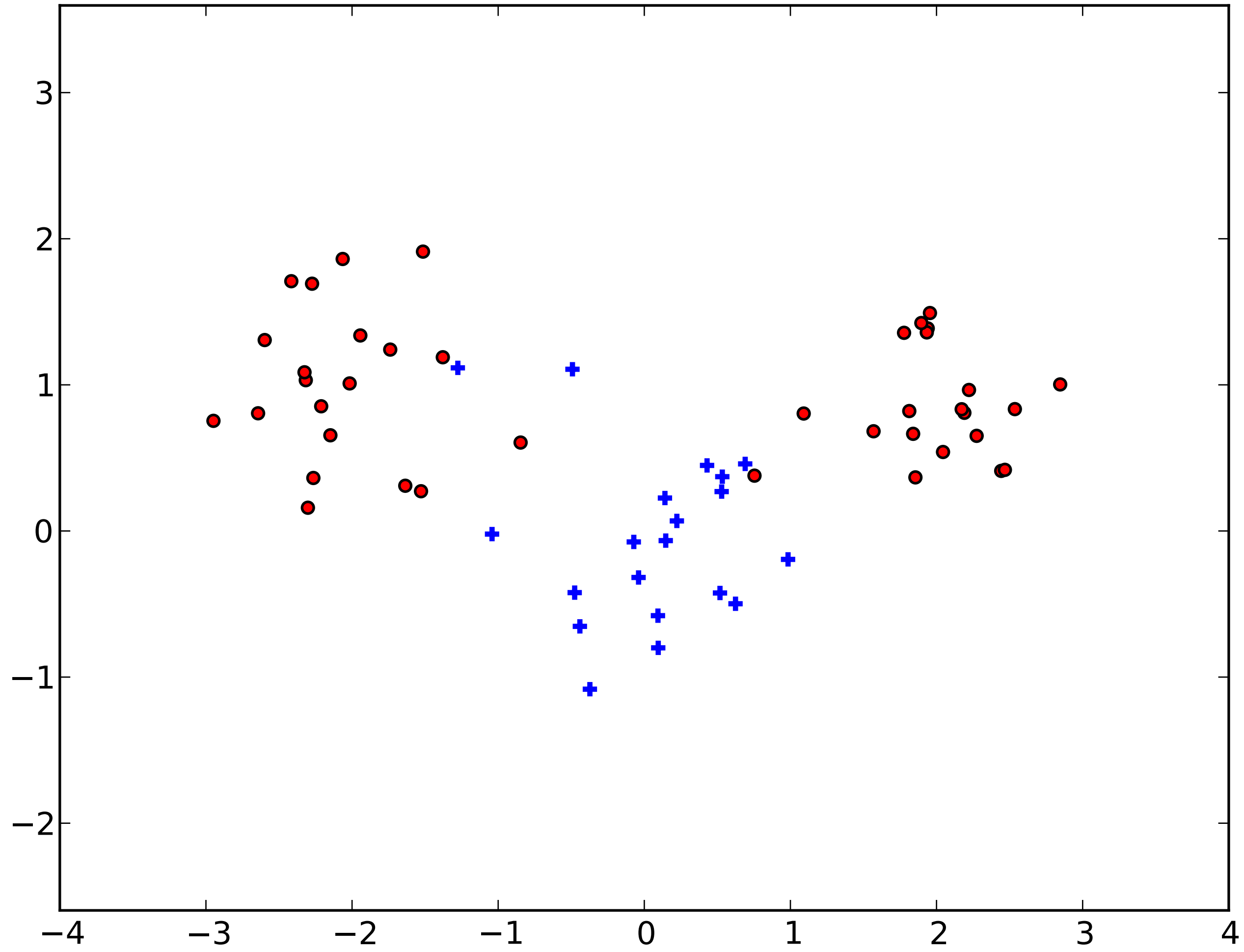
$$K(x, x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$



Radial basis kernel

- The feature vectors can be infinite dimensional… this means that they have unlimited expressive power

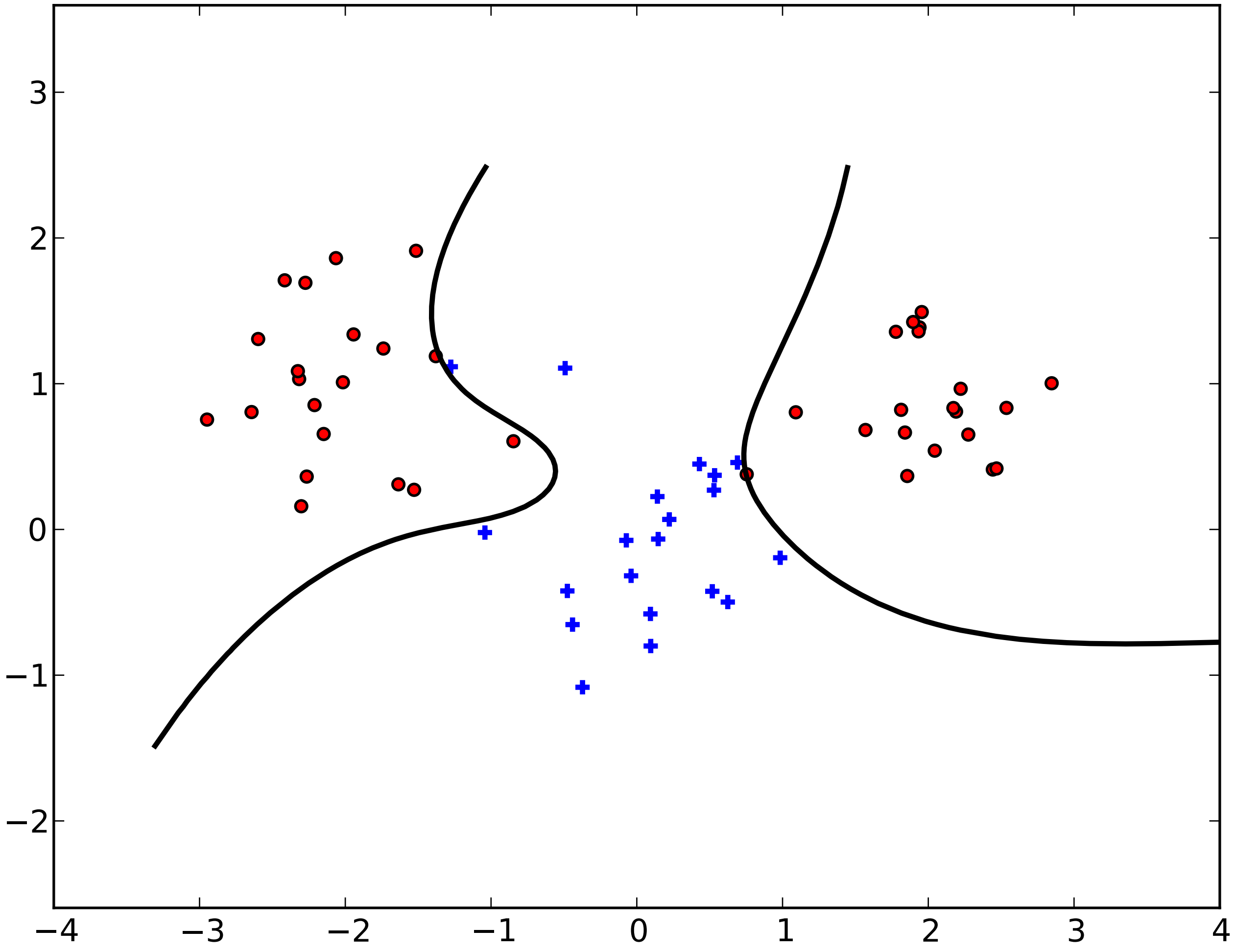
$$K(x,x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$



Radial basis kernel

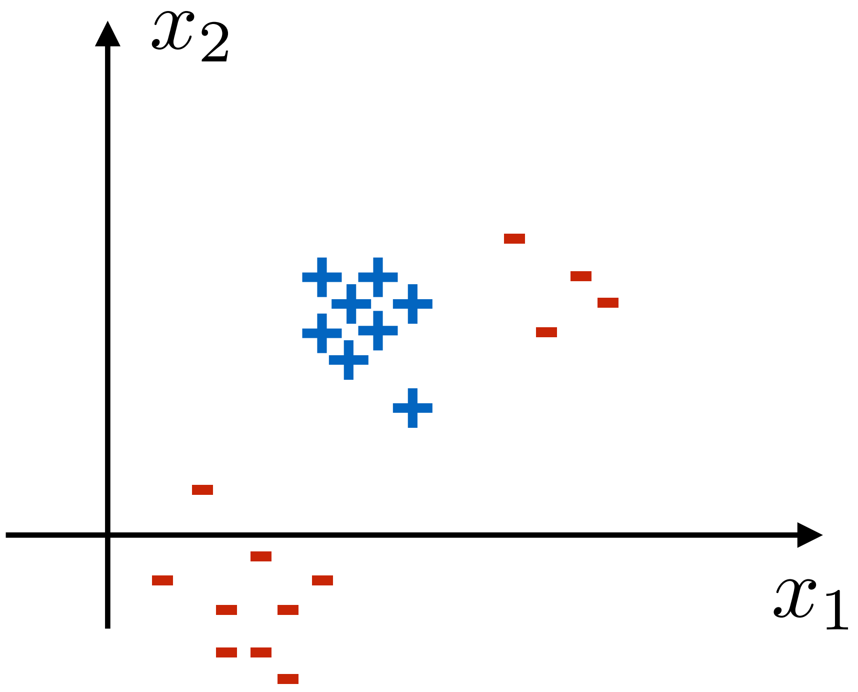
- The feature vectors can be infinite dimensional… this means that they have unlimited expressive power

$$K(x, x') = \exp(-\frac{1}{2}\|x - x'\|^2)$$



Other non-linear classifiers

- Random forest is a good default classifier for (almost) any setting



- Procedure:
  - boostrap sample
  - build a (randomized) decision tree
  - average predictions (ensemble)

Summary

- We can get non-linear classifiers or regression methods by simply mapping examples into feature vectors non-linearly, and applying a linear method on the resulting vectors
- These feature vectors can be high dimensional, however
- We can turn the linear methods into kernel methods by casting the computations in terms of inner products
- A kernel function is simply an inner product between two feature vectors
- Using kernels is advantageous when the inner products are faster to evaluate than using explicit vectors (e.g., when the vectors would be infinite dimensional!)