Machine & Deep Learning

Pr Rahhal ERRATTAHI
rahhal.errattahi@um6p.ma
errattahi.r@ucd.ac.ma

Lecture 04

Kernel Methods

Lecture 4 Overview

- Feature Maps
- Kernel trick
- Kernelization
- Kernelized SVMs
- Summary

Lecture 4 Overview

- Feature Maps
- Kernel trick
- Kernelization
- Kernelized SVMs
- Summary

Limitations of linear models

Linear classifiers cannot deal with

- Non-linearly separable data
- Noisy data
- + this formulation only deals with vectorial data

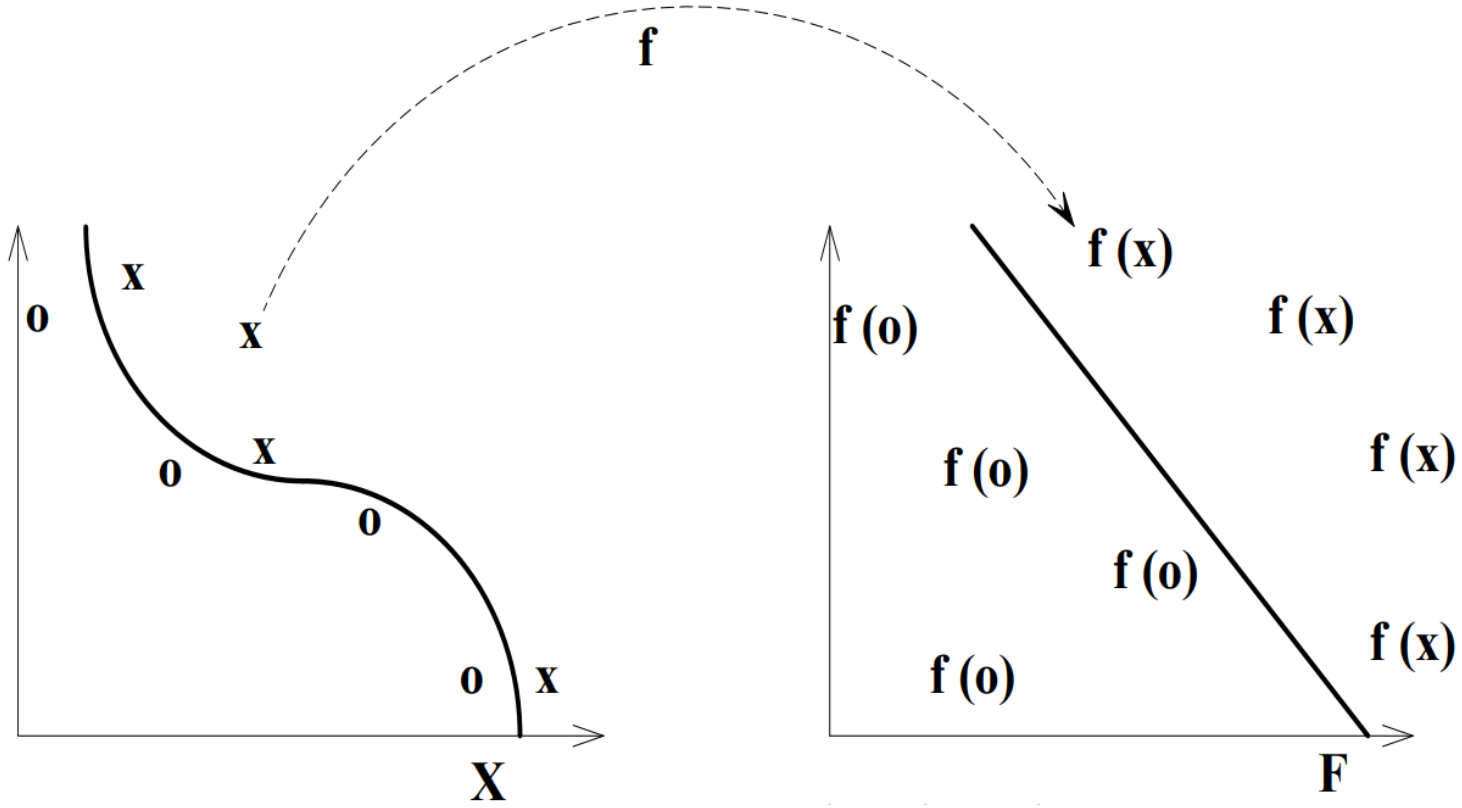
Non-Linear Classifiers

- One solution: creating a net of simple linear classifiers (neurons): a Neural Network (problems: local minima; many parameters; heuristics needed to train; etc)
- Other solution: map data into a richer feature space including non-linear features, then use a linear classifier

Feature Maps

Map data into a feature space where they are linearly separable

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



Feature Maps

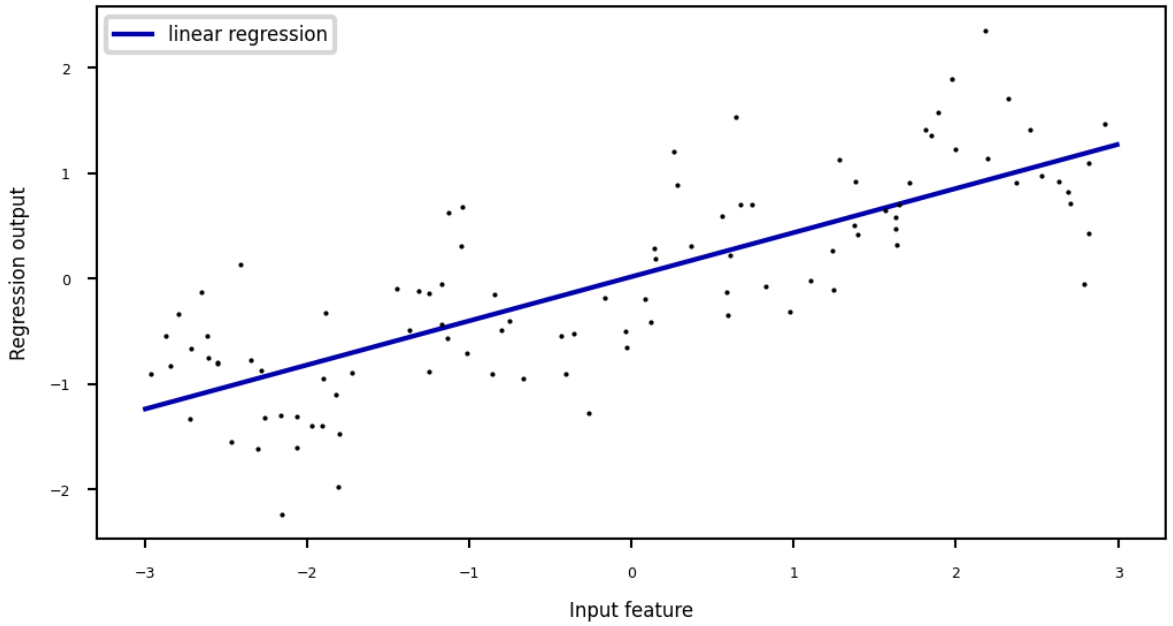
- Linear models: $\hat{y} = W^T x + w_0 = \sum_{i=1}^{P} w_i \cdot x_i + w_0$
- When we cannot fit the data well, we can add non-linear transformations of the features
- Feature map (or *basis expansion*) $\phi : X \rightarrow \mathbb{R}^d$

$$\hat{y} = W^T x \rightarrow \hat{y} = W^T \phi(x)$$

  - E.g. Polynomial feature map: all polynomials up to degree d and all products

$$[1, x_1, \dots, x_p] \rightarrow [1, x_1, \dots, x_p, x_1^2, \dots, x_p^2, \dots, x_p^2, x_1 x_2, \dots, x_{p-1} x_p]$$

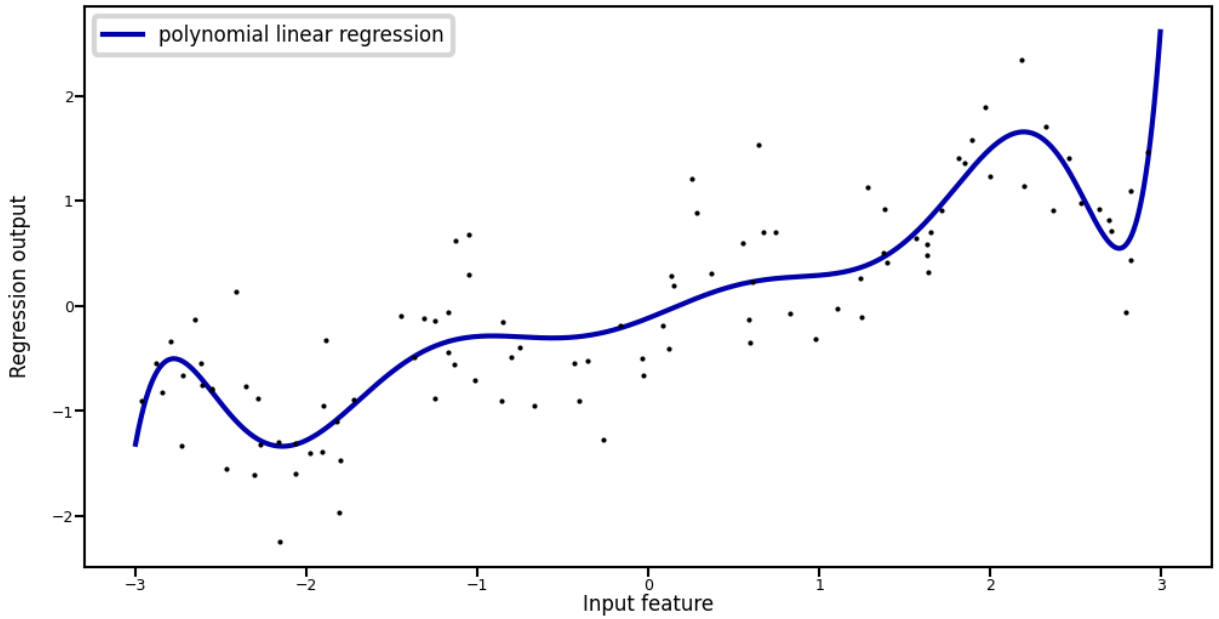
Feature Maps: Ridge regression example



Add all polynomials x^d up to degree 10 and fit again:

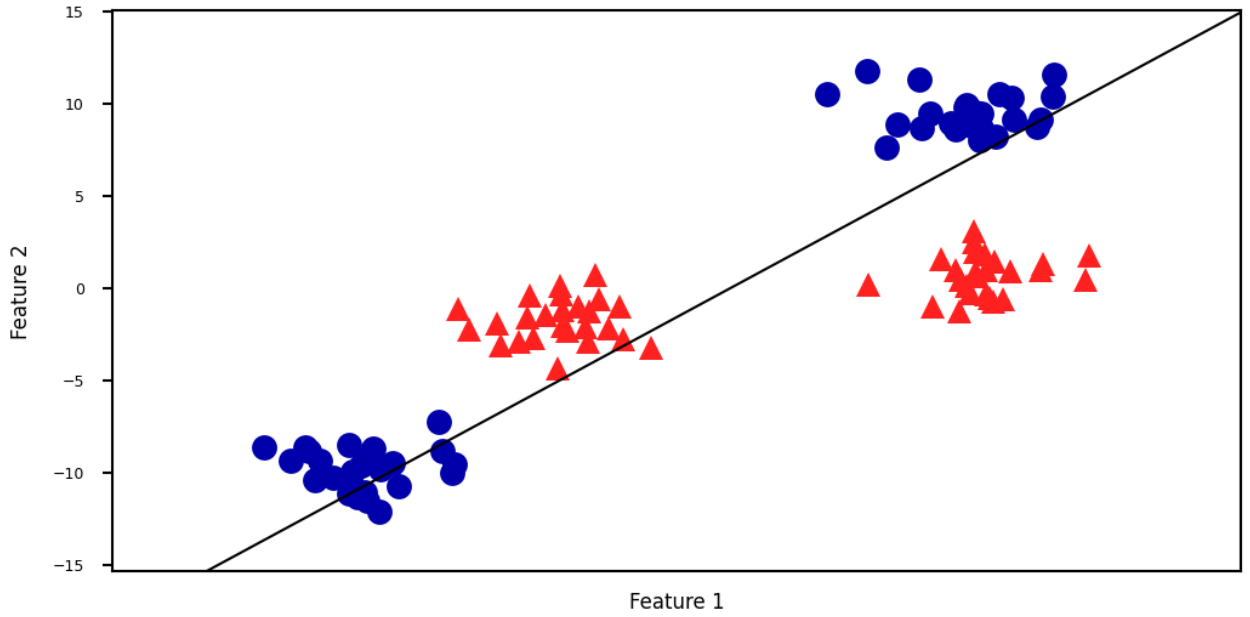
e.g. use sklearn *PolynomialFeatures*

Feature Maps: Ridge regression example



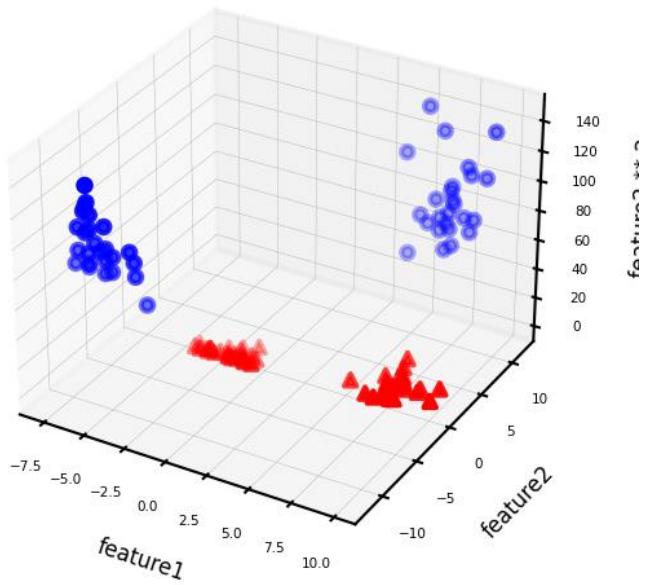
- You may need MANY dimensions to fit the data
  - Memory and computational cost
  - More weights to learn, more likely overfitting

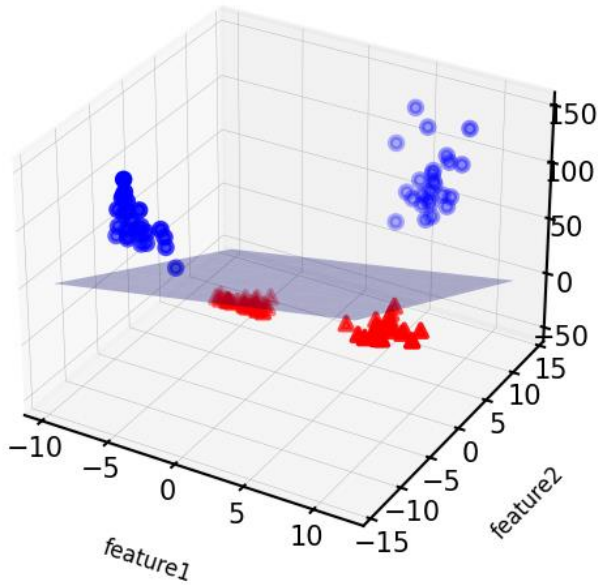
Feature Maps: Linear SVM example



- We can add a new feature by taking the squares of feature1 values

Feature Maps: Linear SVM example





- As a function of the original features, the decision boundary is now a polynomial as well

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_2^2$$

Lecture 4 Overview

- Feature Maps
- Kernel trick
- Kernelization
- Kernelized SVMs
- Summary

Kernel trick

- Computations in explicit, high-dimensional feature maps are expensive
- Kernel trick is a way to use feature maps $\phi : \mathbb{R}^p \rightarrow \mathbb{R}^d$ with linear models but avoid (explicitly) doing the following:
  - represent weight vector $w \in \mathbb{R}^d$
  - compute $\phi(x)$ for any x
- For some feature maps, we can, however, compute distances between points cheaply
  - Without explicitly constructing the high-dimensional space at all

Kernel trick

- Example: quadratic feature map for $x = (x_1, \dots, x_p)$

$$\phi(x) = (x_1, \dots, x_p, x_1^2, \dots, x_p^2, \sqrt{2}x_1x_2, \dots, \sqrt{2}x_{p-1}x_p)$$

- A kernel function exists for this feature map to compute dot products

$$k_{quad}(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) = x_i \cdot x_j + (x_i \cdot x_j)^2$$

- Skip computation of $\phi(x_i)$ and $\phi(x_j)$ and compute $k_{quad}(x_i, x_j)$ directly

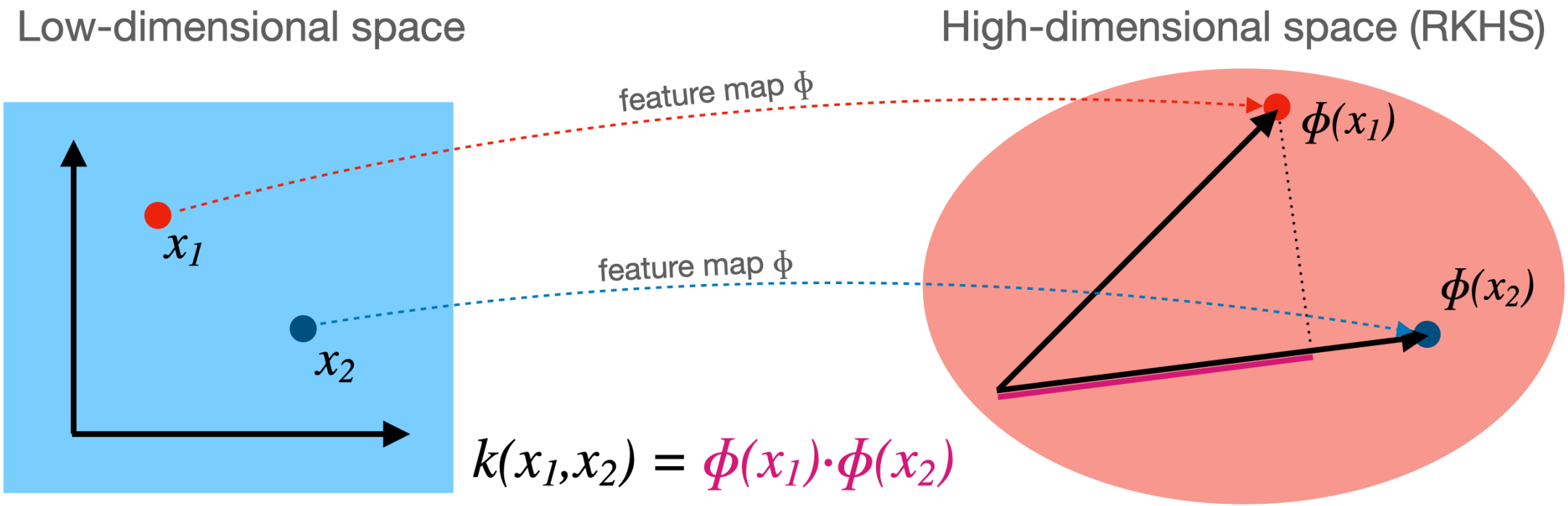
Lecture 4 Overview

- Feature Maps
- Kernel trick
- Kernelization
- Kernelized SVMs
- Summary

Kernelization

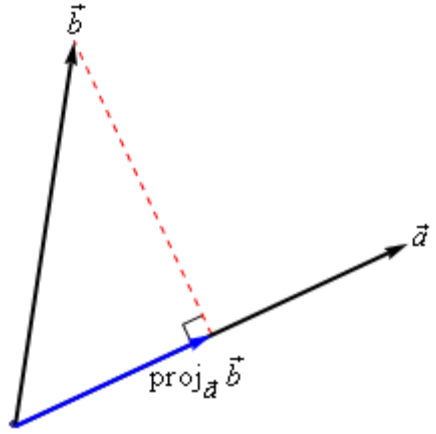
- Kernel k corresponding to a feature map ϕ: $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$
- Computes dot product between x_i, x_j in a high-dimensional space $\mathcal{H}$
  - Kernels are sometimes called *generalized dot products*
  - $\mathcal{H}$ is called the *reproducing kernel Hilbert space* (RKHS)
- The dot product is a measure of the *similarity* between x_i, x_j
  - Hence, a kernel can be seen as a similarity measure for high-dimensional spaces
- If we have a loss function based on dot products x_i, x_j it can be *kernelized*
  - Simply replace the dot products with $k(x_i, x_j)$

Kernelization



Kernelization: Linear kernel

- Input space is same as output space: $X = \mathcal{H} = \mathbb{R}^d$
- Feature map $\phi(x) = \mathrm{x}$
- Kernel: $k_{lenear}(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) = x_i \cdot x_j$
- Geometrically, the dot product is the *projection* of x_j on hyperplane defined by x_i
  - Becomes larger if x_i and x_j are in the same 'direction'



Kernelization: Polynomial kernel

- If k_1, k_2 are kernels, then $\lambda . k_1$ ($\lambda \geq 0$), $k_1 + k_2$, and $k_1 . k_2$ are also kernels
- The **polynomial kernel** (for degree $p \in \mathbb{N}$) reproduces the polynomial feature map

$$k_{poly}(x_i, x_j) = (\gamma(x_i . x_j) + c_0)^{\mathrm{p}}$$

  - γ is a scaling hyperparameter (default $\frac{1}{p}$)
  - c_0 is a hyperparameter (default 1) to trade off influence of higher-order terms

Kernelization: RBF (Gaussian) kernel

- The *Radial Basis Function* (RBF) feature map is related to the Taylor series expansion of e^x

$$\Phi(x) = e^{-x^2/2\gamma^2}\left[1, \sqrt{\frac{1}{1!\gamma^2}}x, \sqrt{\frac{1}{2!\gamma^4}}x^2, \sqrt{\frac{1}{3!\gamma^6}}x^3, \dots\right]^T$$

- RBF (or *Gaussian*) kernel with *kernel width* $\gamma \geq 0$:

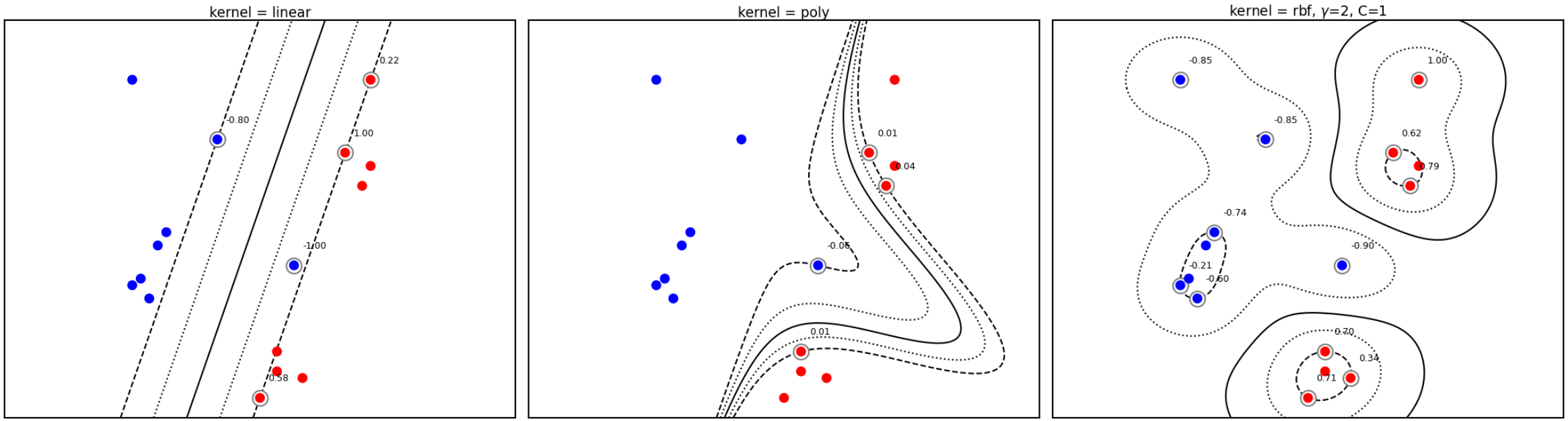
$$k_{RBF}(x_i, x_j) = \exp(-\gamma \left|\left|x_i - x_j\right|\right|^2)$$

Lecture 4 Overview

- Feature Maps
- Kernel trick
- Kernelization
- Kernelized SVMs
- Summary

Kernelized SVMs

- You can use SVMs with any kernel to learn non-linear decision boundaries

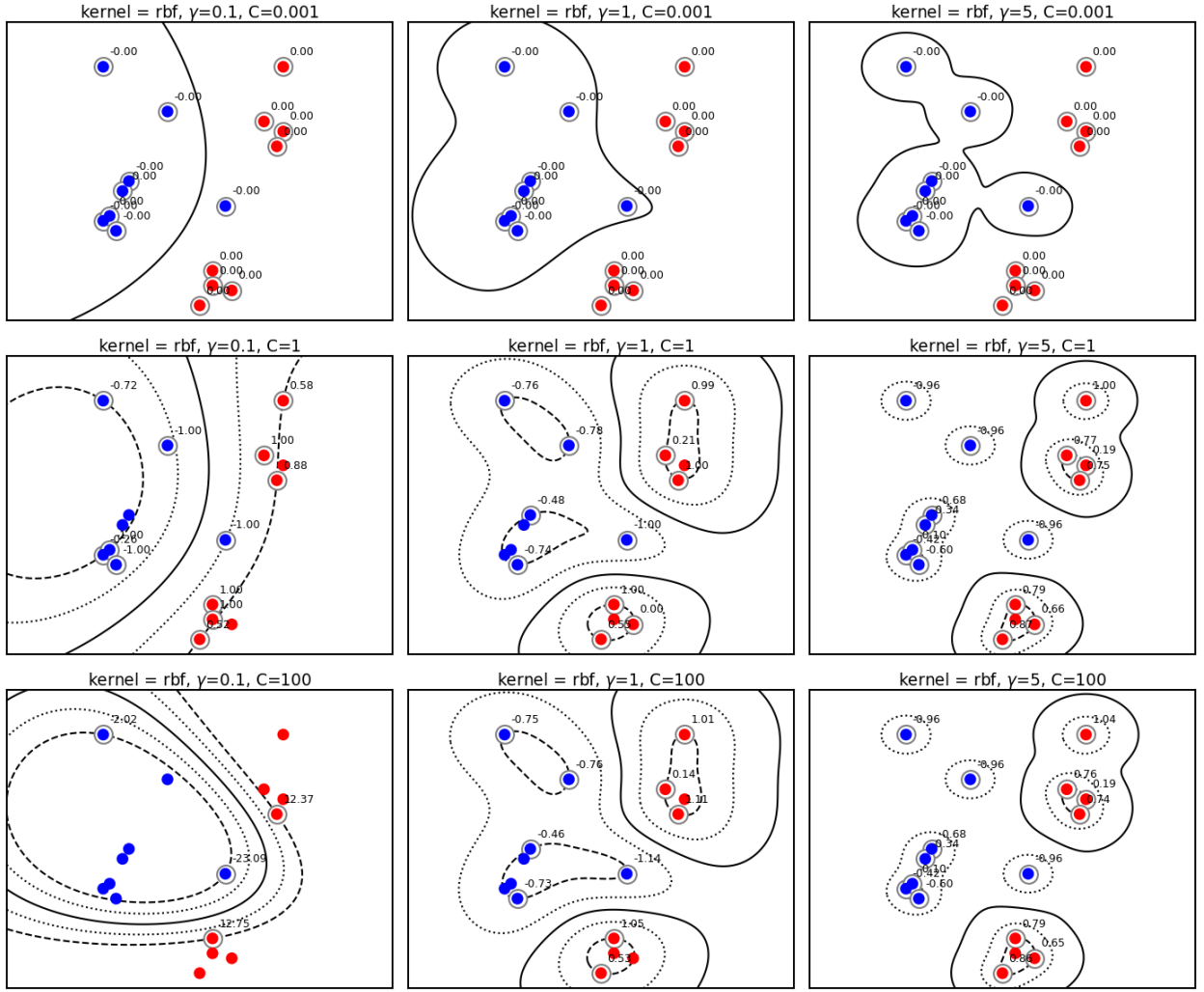


Kernelized SVMs

Tuning RBF SVMs:

- gamma (kernel width)
  - high values cause narrow Gaussians, more support vectors, overfitting
  - low values cause wide Gaussians, underfitting
- C (cost of margin violations)
  - high values punish margin violations, cause narrow margins, overfitting
  - low values cause wider margins, more support vectors, underfitting

Kernelized SVMs



Kernelized SVMs

- C and gamma always need to be tuned
  - Interacting regularizers. Find a good C, then finetune gamma
- SVMs expect all features to be approximately on the same scale
  - Data needs to be scaled beforehand
- Allow to learn complex decision boundaries, even with few features
  - Work well on both low- and high dimensional data
  - Especially good at small, high-dimensional data
- Hard to inspect, although support vectors can be inspected
- In sklearn, you can use SVC for classification with a range of kernels
  - SVR for regression

Lecture 4 Overview

- Feature Maps
- Kernel trick
- Kernelization
- Kernelized SVMs
- Summary

Summary

- Feature maps $\phi(x)$ transform features to create a higher-dimensional space
  - Allows learning non-linear functions or boundaries, but very expensive/slow
- For some $\phi(x)$, we can compute dot products without constructing this space
  - Kernel trick: $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$
  - Kernel k (generalized dot product) is a measure of similarity between x_i and x_j
- There are many such kernels
  - Polynomial kernel
  - RBF (Gaussian) kernel
  - A kernel matrix can be precomputed using any similarity measure (e.g. for text, graphs,…)
- Any loss function where inputs appear only as dot products can be kernelized
  - E.g. Linear SVMs: simply replace the dot product with a kernel of choice

Lab 6 -Kernel Methods