Nonlinear classifiers, kernel methods and SVM

Václav Hlaváč

Czech Technical University in Prague

Czech Institute of Informatics, Robotics and Cybernetics

160 00 Prague 6, Jugoslávských partyzánů 1580/3, Czech Republic

http://people.ciirc.cvut.cz/hlavac, vaclav.hlavac@cvut.cz

also Center for Machine Perception, http://cmp.felk.cvut.cz

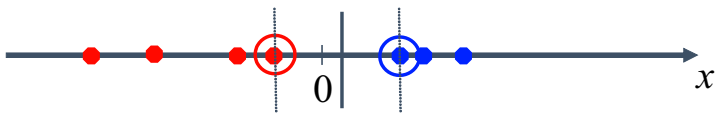
Courtesy: V. Franc.

Outline of the talk:

- Straightening of the feature space.
- Kernel trick, kernel function.
- Selecting kernel function.
- Commonly used kernels.
- Issues with kernel functions.
-

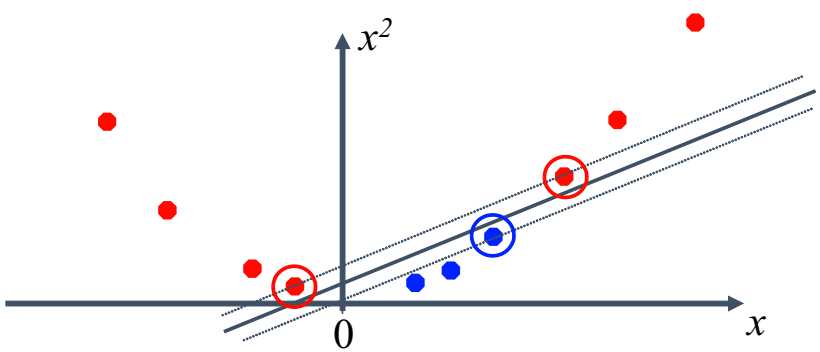
Non-linear decision making tasks, graphical motivation in 1D

- Some data is linearly separable even with some noise. Linear classifiers work well for them.



- Some data is not linearly separable. What can be done?

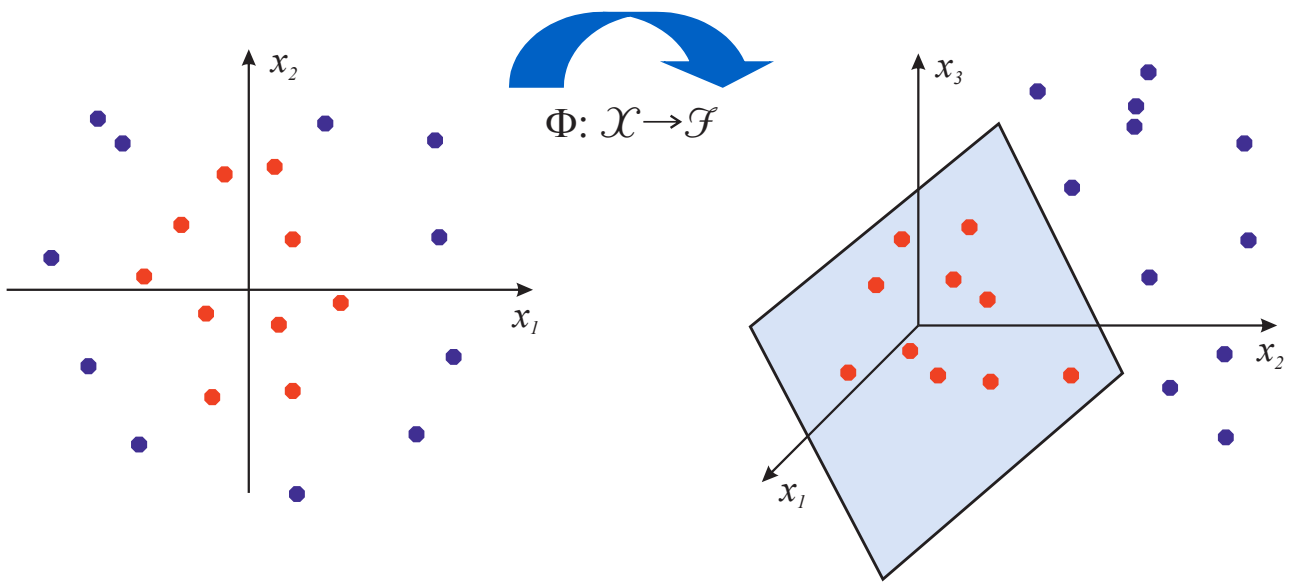
- Mapping the original task to a higher dimensional space helps.



Non-linear decision making tasks, graphical motivation in 2D

A general principle:

The original feature space can be always mapped (straightened) to some higher-dimensional feature space, in which the data is linearly separable.



Naïve feature space straightening

- We learned already about the possibility to explicitly straighten quadratic decision strategies by embedding the problem into a higher dimensional linear space and the linear classifier for them.
- $\Phi\colon \mathcal{X} \to \mathcal{F}$, such as $q'(x, w, b) = w^\top \Phi(x) + b = \sum_{i=1}^{n} w_i \Phi_i(x_i) + b$.
- After the mapping $\Phi(x)$ is used, the decision strategy $q'(x, w, b)$ is linear in $\mathcal{F}$.
- The problem is the excessively high dimension of the obtained linear space, $\dim(\mathcal{F}) \gg \dim(\mathcal{X})$. The original n dimensional nonlinear space is transformed into the $\left(n + \frac{1}{2} n(n+1)\right)$-dimensional feature space.

  Example:

Old dimension	1	2	3	4	5	6	10	20
New dimension	2	5	9	14	20	27	65	230

Problems of a naïve feature space straightening

Two major problems:

1. *Statistical*: The operation in high-dimensional spaces is ill-conditioned due to the 'curse of dimensionality'. There is a risk of overfitting.
2. *Computational*: working in high-dimensions requires higher computational power, which poses limits on the size of the problems that can be tackled.

SVM solution to the problems:

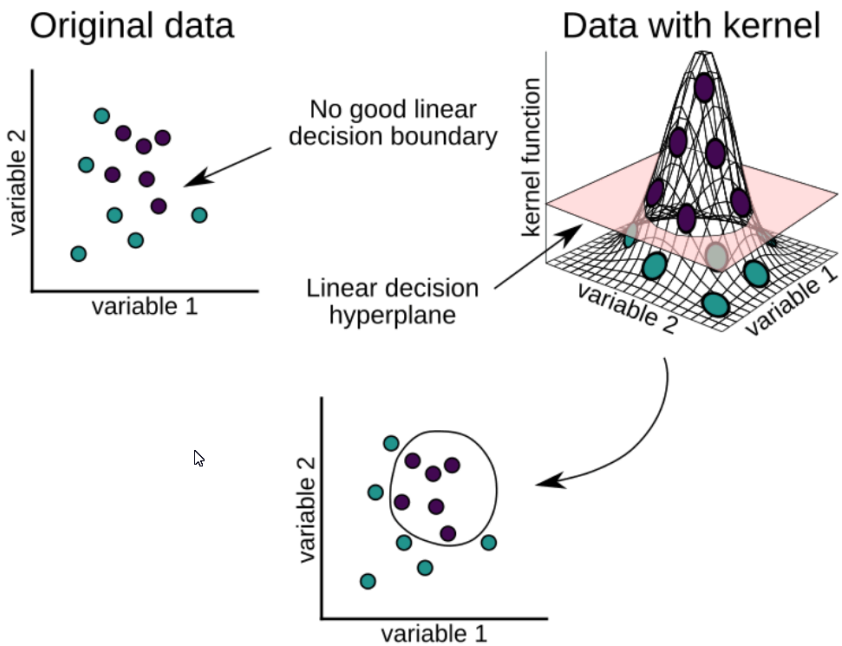
1. Generalization capabilities are ensured in a high-dimensional manifold by enforcing the largest margin classifier. Generalization in SVMs is strictly a function of the margin (or the VC-dimension), regardless of the dimensionality of the feature space.
2. Projection onto a high-dimensional manifold is only implicit because observations from the training set appear as dot products only. Non-linear mapping is realized by kernels.

Kernel trick

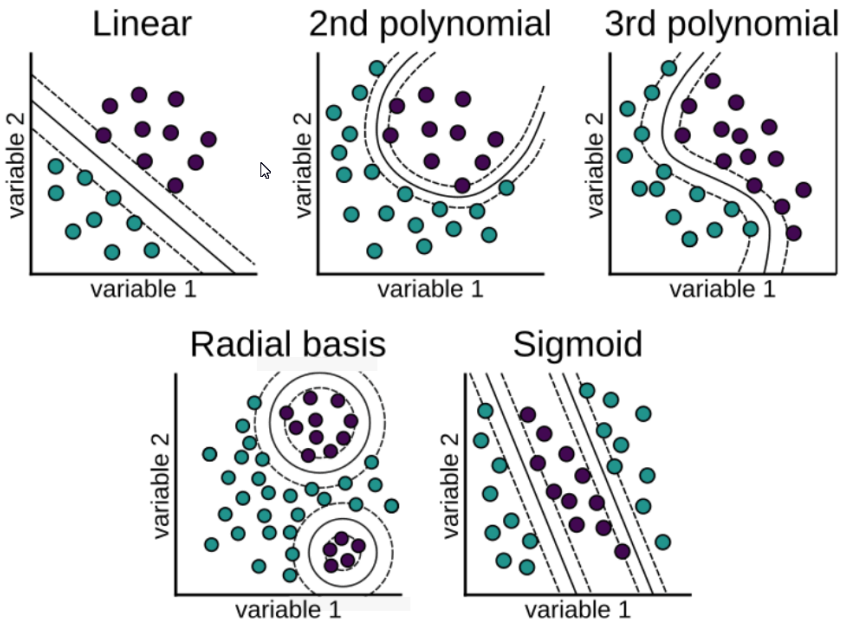
- The 'kernel trick' maps observations from an observation space X into an inner product space V (equipped with its natural norm), without having to compute the mapping explicitly, because the observations will gain a meaningful linear structure in the inner product space.
- The kernel trick allows performing calculations more efficiently if the classification algorithm uses observations x only in a form of a dot product.
- Recall: SVM classifier provides dual-task formulation in which data points x appear in the form of a dot product (as in SVMs).

- Kernel trick was suggested first in 1964 by M. Aizerman, E. Braverman, and L. Rozonoer, *Theoretical foundations of the potential function method in pattern recognition learning.* Automation and Remote Control 25: 821–837.
- *Notation: Kernel functions are usually denoted by K in the literature. We denote kernel functions by the Greek letter κ (kappa) because our book Schlesinger, Hlavac 2002 uses K for hidden states.*

Motivation; Kernel for a local feature space straightening



Adding one extra dimension, here stretching into 3D, makes data points linearly separable.



Kernel functions, examples. Solid line = decision boundary (projected back onto the original feature space). Dashed lines = margin.

Courtesy: Figures from the book Hefin I. Rhys: Machine Learning with R, the tidyverse, and MLR. March 2020. Manning Publications Co., USA

Kernel functions

- Kernel functions $\kappa\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are symmetric and positive-definite. κ is often interpreted as a similarity measure.
- Kernel function $\kappa(x_1, x_2) \doteq \Phi^\top(x_1)\,\Phi(x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$, i.e., the kernel function equals to the scalar product of the non-linearly mapped original features.
- Kernel function $\kappa(x_1, x_2)$ can be evaluated without explicit mapping $\Phi\colon \mathcal{X} \to \mathcal{F}$.
- If one is insightful regarding a particular machine learning problem, one may manually construct the kernel function κ and verify it.
- An explicit representation for Φ is not required. It suffices to know that it maps into the inner product space V. Conveniently, based on Mercer's theorem (We will introduce it soon.), it suffices to equip the original space X with one's choice of measure and verify that $\kappa\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfies Mercer's condition.
- The benefit: the infinite-dimensional mapping is implemented. Nevertheless, the calculations are in a finite dimension.

SVM, non-separable linearly with kernels

- Transition to the dual task and related optimization using Lagrange coefficients α_i, $i = 1 \dots L$, is analogical to the linear non-separable case.
- The dot product $x_i^\top x_j$ is replaced by the kernel $\kappa(x_i, x_j)$.
- The optimization problem reads now

$$\alpha_i = \underset{\alpha_i}{\operatorname{argmax}} \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i=1}^{L} \sum_{j=1}^{L} \alpha_i \, \alpha_j \, y_i \, y_j \, \kappa(x_i, x_j) \,, \quad 0 \leq \alpha_i \leq C \,, \quad \sum_{i=1}^{L} \alpha_i y_i = 0 \,.$$

- The decision strategy is

$$q(x) = w^\top \Phi(x) + b = \sum_{i=1}^{L} \alpha_i \, y_i \, \kappa(x_i, x) + b \,.$$

Mercer's theorem

- In mathematics, Mercer's theorem has implications in the theory of integral equations (James Mercer 1909).
- Mercer's condition tells us whether a candidate kernel is actually an inner-product kernel in some space.
- Let $\kappa(x_1, x_2)$ be a continuous symmetric kernel defined in the closed interval $a \leq x \leq b$. The kernel can be expanded into series $\sum_{i=1}^{\infty} \lambda_i \phi(x_1)\phi(x_2)$. Functions ϕ reside in Hilbert space, which is a 'generalization' of Euclidean space. Here the inner product can be any inner product, not just its special case, the common dot product.
- Consider $\lambda_i > 0$, $\forall i$. The necessary and sufficient condition for the uniform convergence of the series $\sum_{i=1}^{\infty} \lambda_i \phi(x_1)\phi(x_2)$ is that $\int_a^b \int_a^b \kappa(x_1, x_2)\, \phi(x_1)\, \phi(x_2)\, \mathrm{d}x_1\, \mathrm{d}x_2 \geq 0$ holds for all $\phi(.)$, for which $\int_a^b \phi^2(x)\, \mathrm{d}x < \infty$. It can be understood as a generalization of the semidefiniteness condition for a matrix.

How do we select a kernel function?

- We saw the expansion

$$\sum_{i=1}^{\infty} \lambda_i \, \phi(x_1) \, \phi(x_2)$$

  in the previous slide. Functions $\phi_i(\cdot)$ are called eigenvectors in the expansion.
- Numbers λ_i are eigenvalues.
- The fact that all of the eigenvalues are nonnegative means that the kernel is positive semidefinite.
- Notice that the dimensionality of the implicit space can be infinitely large.
- Mercer's condition only tells us whether a kernel is actually an inner-product kernel. It does not tell us how to construct the functions $\phi_i(x)$ for the expansion.

Mercer's theorem in a simplified form

- In our simpler setting dealing with the observation space X, one can use the counting measure $\mu(T) = |T|$ for all $T \subset X$.
- In such a setting, the integral in Mercer's theorem reduces to a simple summation

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \kappa(x_i, x_j)\, c_i c_j \geq 0$$

for all finite sequences of points $x_1, \ldots, x_n \in X$ and all choices of real numbers $c_1, \ldots, c_n$ (recall the positive semidefinite kernel).

Kernels complying to Mercer's condition

- Polynomial kernels of degree p, $\kappa(x_1, x_2) = \left(x_1^\top x_2 + c\right)^p$.
- Radial basis functions (also Gaussian kernels)

$$\kappa(x_1, x_2) = \exp\left(\frac{-1}{2\sigma^2}\left\|x_1 - x_2\right\|^2\right) .$$

  The width of the kernel σ is a user defined parameter. The number of radial basis functions is determined automatically.

- Two layer perceptron $\kappa(x_1, x_2) = \tanh(\beta_0\, x_1^\top x_2 + \beta_1)$
  (also saturating, sigmoid-like)
  The number of hidden neurons and their weight vectors are determined automatically by the number of support vectors and their values, respectively. The hidden-to-output weights are the Lagrange multipliers α_i. However, this kernel will only meet Mercer's condition for certain values of β_0 and β_1.

Polynomial kernel, n-dim; derivation for the quadratic kernel

- Polynomial kernel function of degree p; definition in the n-dimensional feature space $x_1, x_2 \in \mathbb{R}^n$, $\kappa(x_1, x_2) = (x_1^\top x_2 + c)^p = \langle \Phi(x_1), \Phi(x_2) \rangle$.
- We substitute $a = x_1; b = x_2$ to prevent nested indices.
- A special case: a quadratic polynomial kernel

$$\kappa(x_1, x_2) = \kappa(a, b) = \left(\sum_{i=1}^{n} a_i b_i + c \right)^2 =$$

$$= \left(\sum_{i=1}^{n} a_i b_i \right)^2 + 2 \sum_{i=1}^{n} a_i b_i \, c + c^2 = \left(\sum_{i=1}^{n} a_i b_i \right)^2 + 2 \sum_{i=2}^{n} \sum_{j=1}^{i-1} a_i a_j b_i b_j + 2 \sum_{i=1}^{n} a_i b_i \, c + c^2 =$$

$$= \sum_{i=1}^{n} a_i^2 b_i^2 + \sum_{i=2}^{n} \sum_{j=1}^{i-1} (\sqrt{2} a_i a_j)(\sqrt{2} b_i b_j) + \sum_{i=1}^{n} (\sqrt{2c} a_i)(\sqrt{2c} b_i) + c^2 = \langle \Phi(a), \Phi(b) \rangle$$

$$\Phi(a) = \left(a_1^2, \ldots, a_n^2, \sqrt{2}\, a_2 a_1, \ldots, \sqrt{2}\, a_n a_{n-1}, \sqrt{2c}\, a_1, \ldots, \sqrt{2c}\, a_1, \ldots, \sqrt{2c}\, a_n, c \right)^\top$$

$$\Phi(b) = \left(b_1^2, \ldots, b_n^2, \sqrt{2}\, b_2 b_1, \ldots, \sqrt{2}\, b_n b_{n-1}, \sqrt{2c}\, b_1, \ldots, \sqrt{2c}\, b_1, \ldots, \sqrt{2c}\, b_n, c \right)^\top$$

Courtesy: Marko Sahan

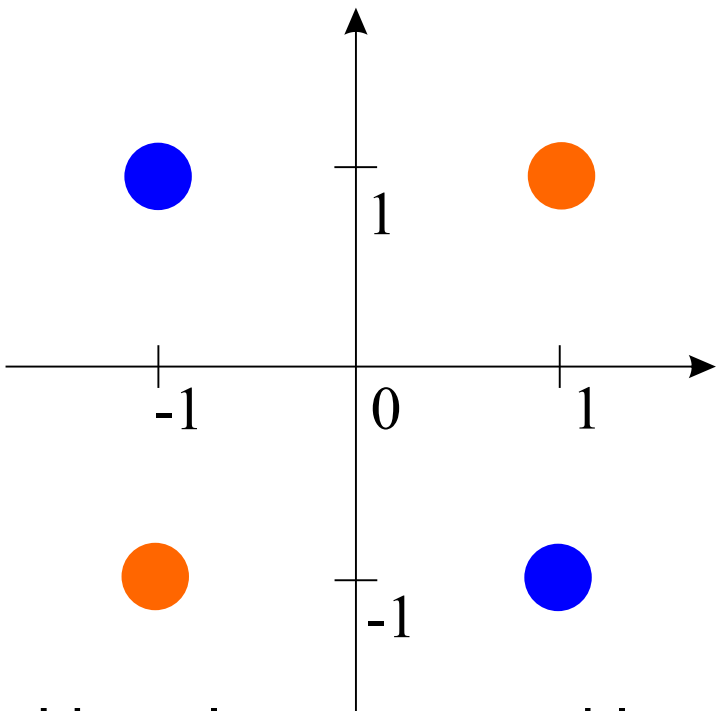
A more special case; a quadratic kernel in $\mathbb{R}^2$

$$\kappa(a,b) = (a_1b_1 + a_2b_2 + 1)^2 = a_1^2\,b_1^2 + a_2^2\,b_2^2 + 2\,a_1a_2b_1b_2 + 2\,a_1b_1 + 2\,a_2b_2 + 1 =$$

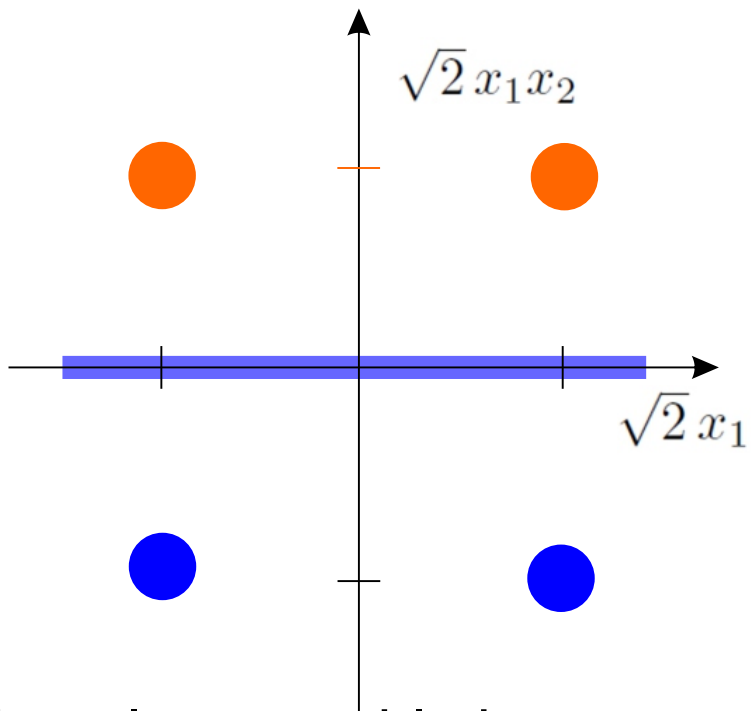
$$= \left\langle \begin{bmatrix} a_1^2 \\ a_2^2 \\ \sqrt{2}\,a_1a_2 \\ \sqrt{2}\,a_1 \\ \sqrt{2}\,a_2 \\ 1 \end{bmatrix}, \begin{bmatrix} b_1^2 \\ b_2^2 \\ \sqrt{2}\,b_1b_2 \\ \sqrt{2}\,b_1 \\ \sqrt{2}\,b_2 \\ 1 \end{bmatrix} \right\rangle$$

Courtesy: Mehryar Mohri

Solution to XOR task with a quadratic kernel

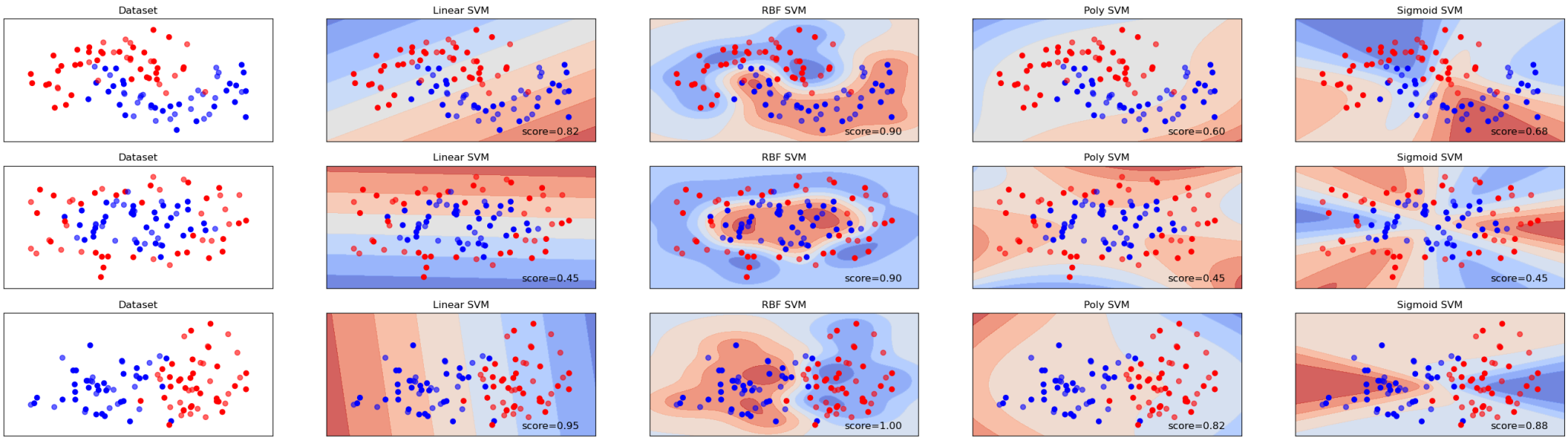


Linearly non-separable



Linearly separable by $x_1x_2 = 0$

Examples using different kernels on three datasets



Courtesy: The examples and plots are modified from WittmannF Python example.

Problems with kernel functions

With SVMs, the kernel methods yield the solution in the form

$$q(x) = \Phi^\top(x) \underbrace{\sum_{i=1}^{L} \alpha_i \, y_i \, \Phi(x_i)}_{w \in \mathcal{F}} + b = \sum_{i=1}^{L} \alpha_i \, y_i \, \kappa(x, x_i) + b \,,$$

where $(x_1, \ldots, x_L)$ is the sequence of vectors in the training multi-set.

- The decision function $q(x) = \sum_{i=1}^{L} \alpha_i \, y_i \, \kappa(x, x_i) + b$ is not sparse, i.e., a lot of α_i are non-zero $\Rightarrow$ slow evaluation.
- Representation of the training set in terms of a dot product is memory demanding for large L since the full kernel matrix $\mathbb{Y}_{i,j} = \kappa(x_i, x_j)$ has dimension $L \times L$.
- Evaluation of $\kappa(x_i, x_j)$ can be computationally demanding.

Notes on SVM performance

- SVMs work very well in practice.
- The solution is sparse when dealing with large data sets. Only support vectors specify the separating hyperplane.
- Large feature spaces can be handled. The complexity does not depend on the feature space dimension.
- Generalization (overfitting) can be controlled by the soft margin approach.
- The user chooses the kernel function, its parameters and the regularization constant C. The rest is automatic.
- SVMs can be expensive in time and space for large data sets.
  - The computation of the maximum-margin hyper-plane has a lower bound $\mathcal{O}(L^2)$ for the nonlinear case and $\mathcal{O}(L)$ for the linear case.
  - All the support vectors have to be stored in a memory.