Support Vector Machines III

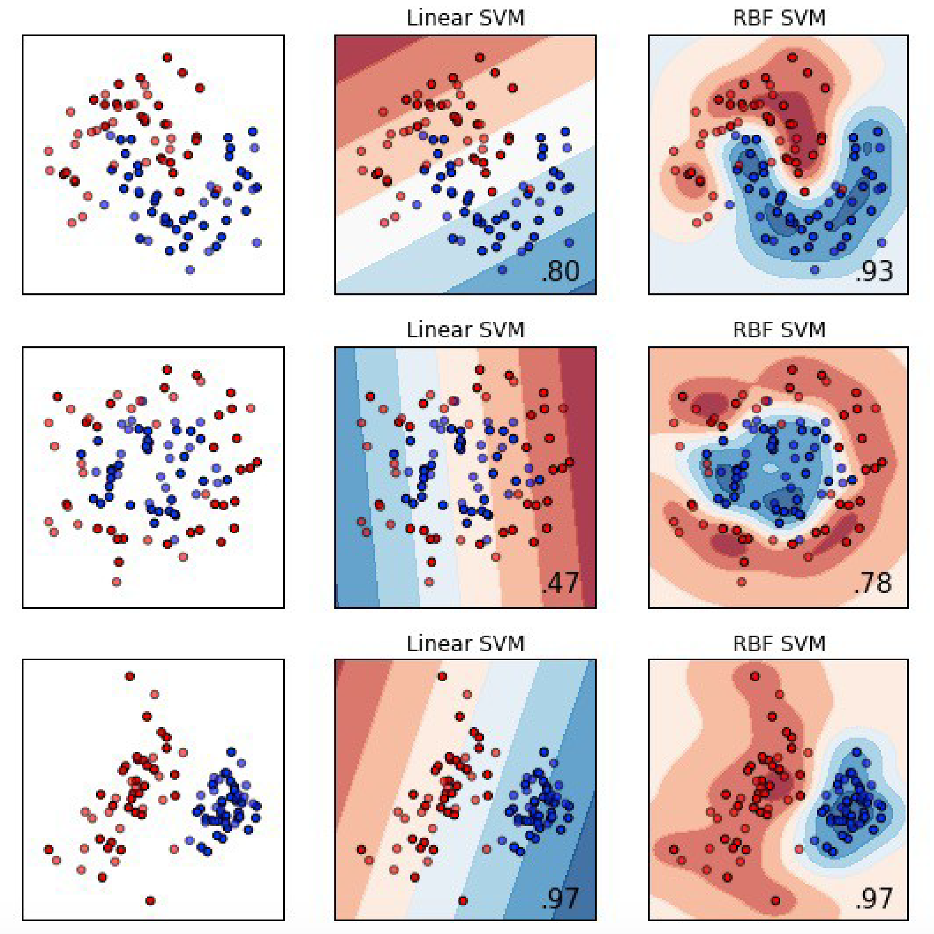
COMPCSI 762

WEEK 10

Instructor: Thomas Lacombe
Based on slides from Meng-Fen Chiang

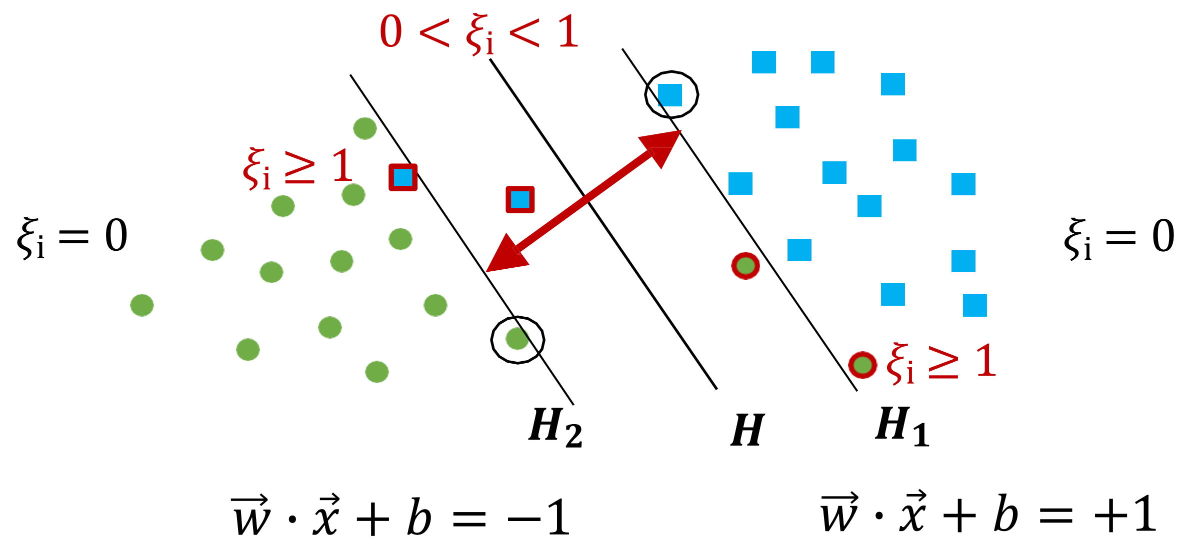
OUTLINE

- Data Characteristics
  - Linearly Separable Data
  - Non-Linearly separable Data
- SVM
  - Linearly Separable Data: Hard-margin SVMs (9.1)
  - Non-Linearly Separable Data: Soft-margin SVMs (9.2)
  - Non-Linearly Separable Data: Kernelized SVMs (9.3)
- Summary



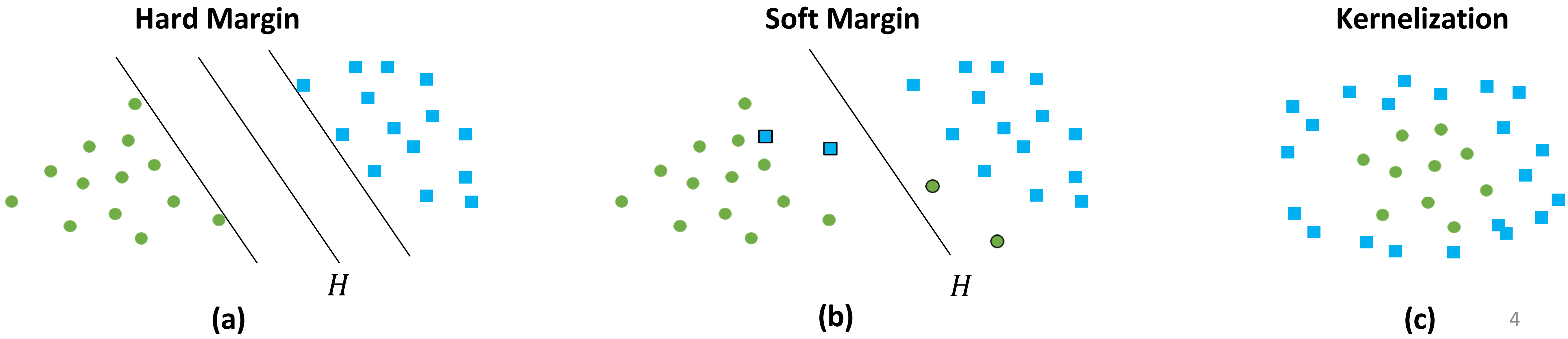
RECAP: Soft-margin Maximization

- Given a set of training data $S = \left((x_1, y_1), \ldots, (x_n, y_n)\right), y_i \in \{+1, -1\}$
- Goal: The soft-margin SVM algorithm aims to find a linear classifier that
    1. Maximizes (γ) the margin on S and
    2. Minimize the misclassification error $C \sum_{i=1}^{n} \xi_i$

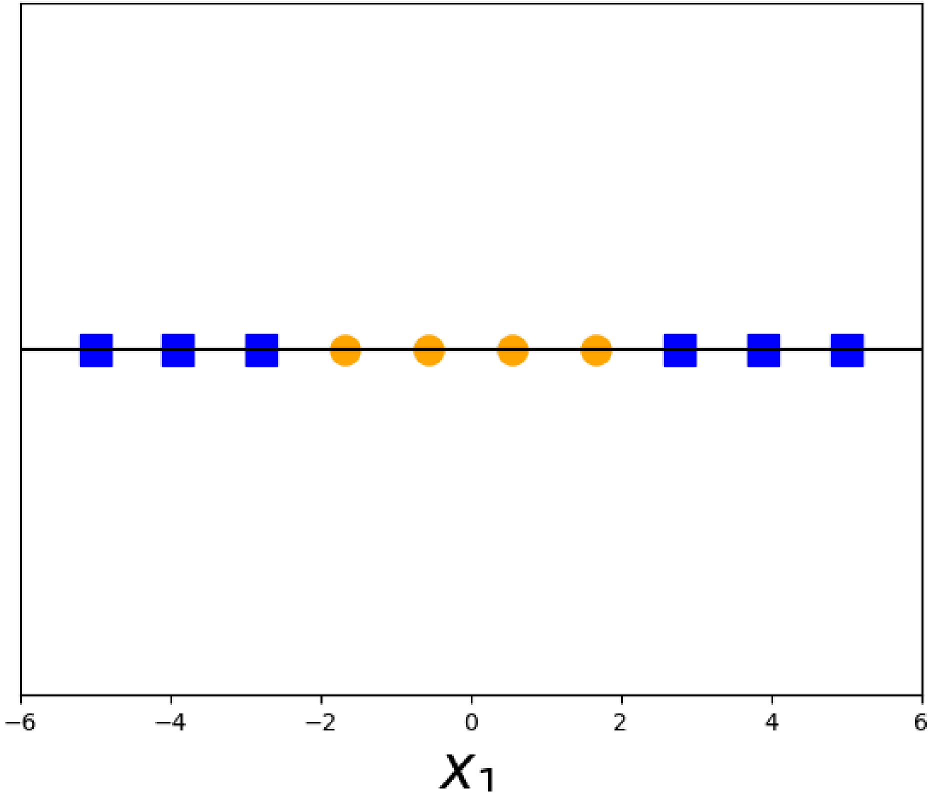


RECAP: Non-Linearly Separable Data

- Input data is not linearly separable in a two-dimensional space
  - Soft-margin SVMs: a linear separating hyperplane that allows misclassifications
  - Kernelized SVMs: non-linear classifier that is a linear separating hyperplane(s) in higher feature space

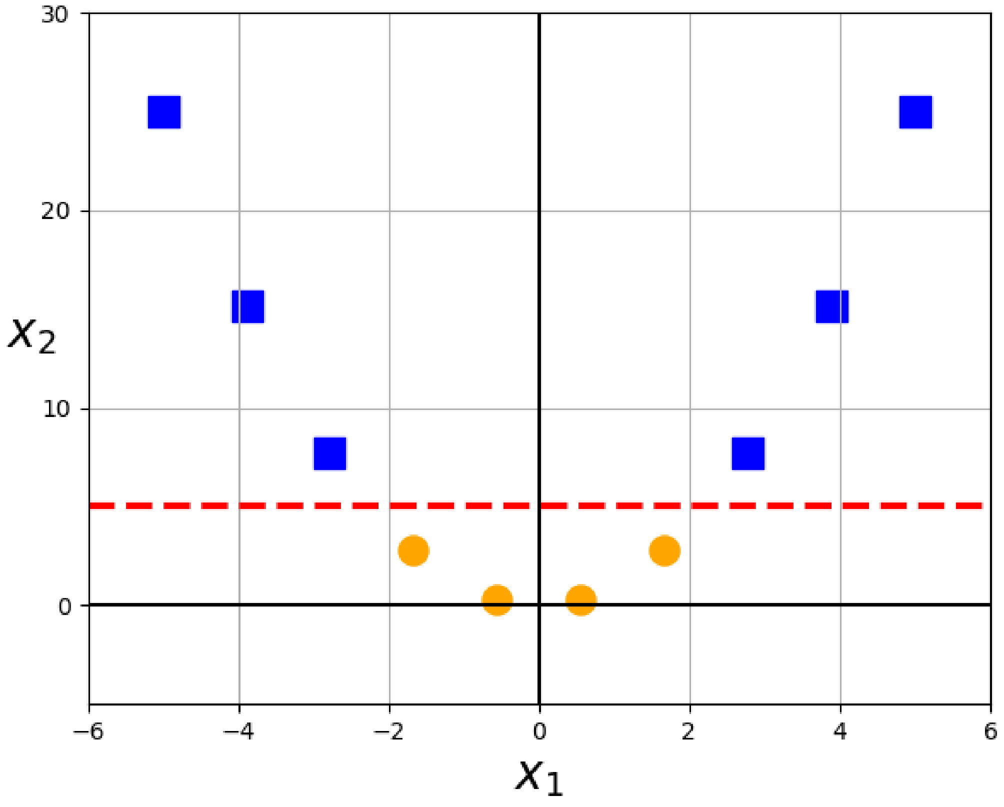


Intuitive example 1



Non-linearly separable data

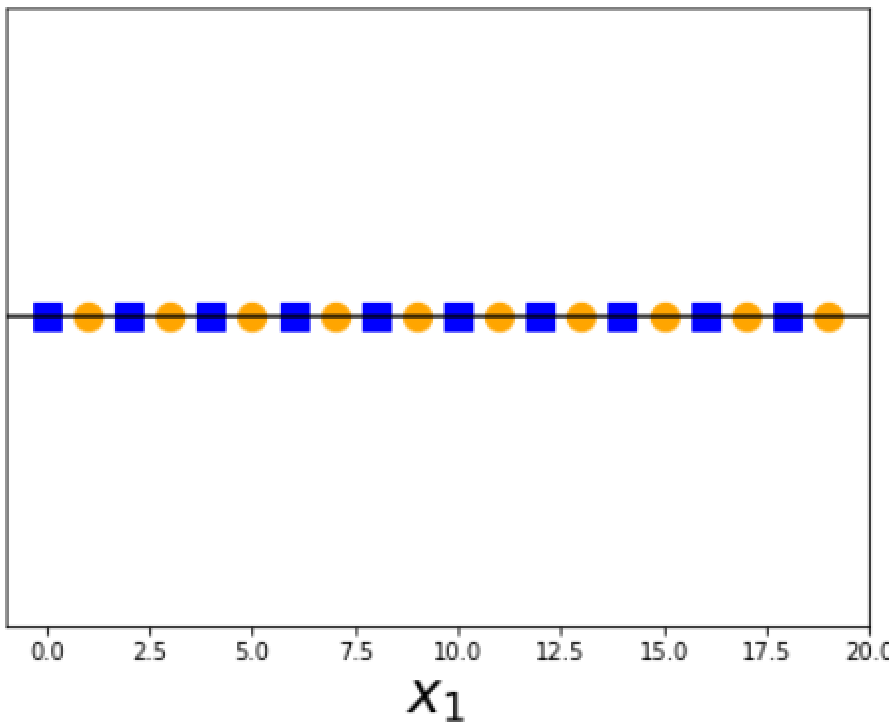
$$\Phi(x) = x^2$$



Linearly separable data!

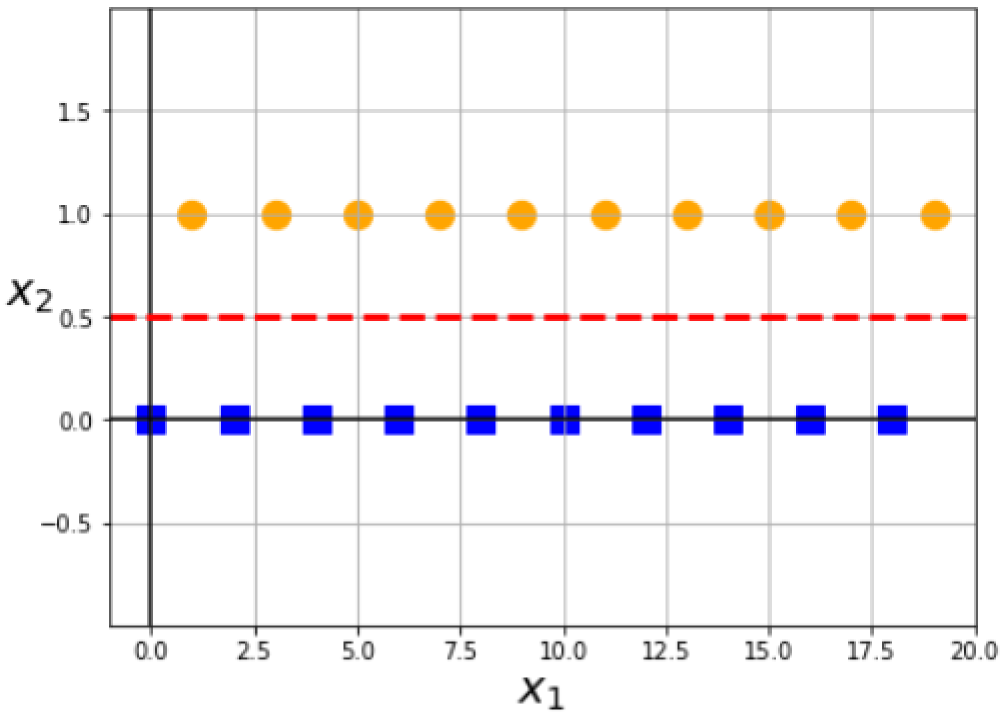
Images source: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

Intuitive example 2



Non-linearly separable data

$$\Phi(x) = x \, mod(2)$$

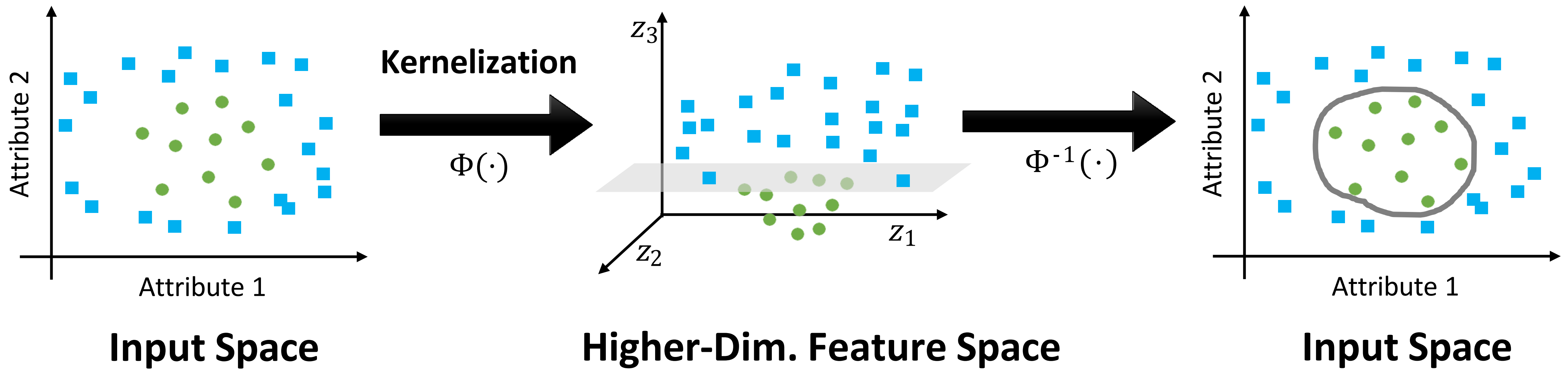


Linearly separable data!

Images source: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

Notion of Feature Transformation

- Non-linearly separable in input data space
- Linearly separable in a ***higher dimensional*** feature space



Problem Definition: Kernelized SVMs

- Given a set of training data $S = \big((x_1, y_1), \ldots, (x_n, y_n)\big), y_i \in \{+1, -1\}$
- Goal: The kernelized SVM algorithm aims to find a non-linear classifier, which is a separating hyperplane(s) in a higher dimensional space

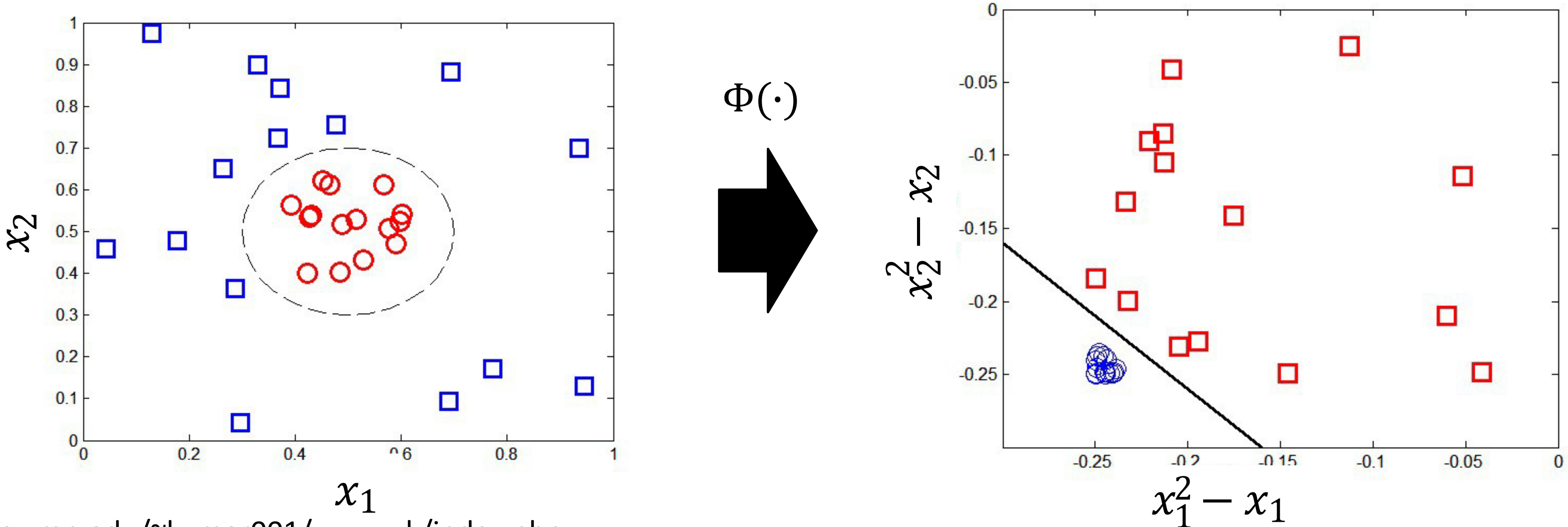
- Approach
    1. Transform the original input data into a higher dimensional space
    2. Search for a linear separating hyperplane in the new space

Step1: Space Transformation

- $\Phi(\cdot)$: Transform the original input data into a higher dimensional space

  $\Phi:(x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2)$

  - Hyperplane: $\vec{w} \cdot \Phi(\vec{x}) + b = 0 \rightarrow w_4 x_1^2 + w_3 x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + b = 0$



Source: https://www users.cs.umn.edu/~kumar001/dmbook/index.php

Step2: Search for a Linear Separating Hyperplane

- Dual Optimization Problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_\mathrm{i} \, y_j \overrightarrow{x_i} \cdot \overrightarrow{x_j} \qquad \text{s.t. } \alpha_i \geq 0 \text{ and } \textstyle\sum_{i=1}^{n} \alpha_i y_\mathrm{i} = 0, \qquad i = 1, 2, \dots, n$$

- Kernelized Optimization Problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_\mathrm{i} \, y_j \Phi(\overrightarrow{x_i}) \cdot \Phi(\overrightarrow{x_j}) \qquad \text{s.t. } \alpha_i \geq 0 \text{ and } \textstyle\sum_{i=1}^{n} \alpha_i y_\mathrm{i} = 0, \qquad i = 1, 2, \dots, n$$

  - Same set of equations as dual problem optimization except that involve $\Phi(\vec{x})$ in feature space, instead of $\vec{x}$ in input space

Step2: Search for a Linear Separating Hyperplane

- Training: Solve the Kernelized Optimization Problem by Quadratic Programming (QP)

$$\vec{w} = \sum_{x_i \in SV} \alpha_i y_\mathrm{i} \Phi(\vec{x_i})$$

$$\vec{b} = \frac{1}{|SV|} \sum_{x_i \in SV} y_i - (\vec{w} \cdot \Phi(\vec{x_i}))$$

- Testing: Determine the class label for a test point $\vec{z}$ by using the learned kernelized SVM ($\vec{w}$ and $\vec{b}$) with support vectors (SV)

$$f(\vec{z}) = sign(\vec{w} \cdot \Phi(\vec{z}) \ + \ b)$$

$$= sign(\sum_{x_i \in SV} \alpha_i y_\mathrm{i} (\Phi(\vec{x_i}) \cdot \Phi(\vec{z})) \ + \ b)$$

Kernel Trick

- Kernelized Optimization Problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_\mathrm{i} y_j \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$$

- In practice, applying the transformation Φ to the original data is often impractical (high number of features and Φ can involve polynomial combinations).

- **Kernel Trick:** Instead of transforming the input data, and then applying the inner product between pairs of transformed data points, a **Kernel function** is applied:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_\mathrm{i} y_j \mathrm{K}(\vec{x_i}, \vec{x_j})$$

Kernel Functions

- **Kernel function:** A kernel function K takes vectors in the original space as inputs, and returns the inner product of the vectors in the transformed space:

$$K\left(\overrightarrow{x_i}, \overrightarrow{x_j}\right) = \Phi(\overrightarrow{x_i}) \cdot \Phi(\overrightarrow{x_j})$$

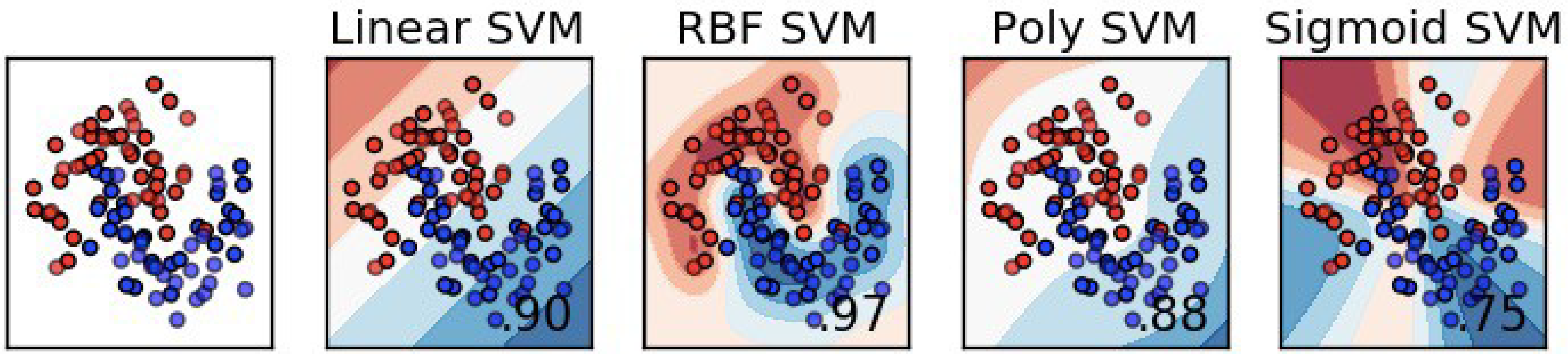
- A kernel function calculates the similarity between pairs of data points.

- Applying the function is equivalent to calculating the inner product in the transformed space, but Φ is never directly calculated (no need to know Φ explicitely).

- A valid kernel function is a function that can be used to compute the inner product between two feature vectors in a high-dimensional space without explicitly mapping them.

Kernel Functions

- There exist a lot of possible kernel functions:



Source: https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization

Kernel Function: Polynomial Kernel

- Polynomial Kernel of Degree d:

$$k(\vec{x},\vec{z}) = (\vec{x}\cdot\vec{z})^{d}$$

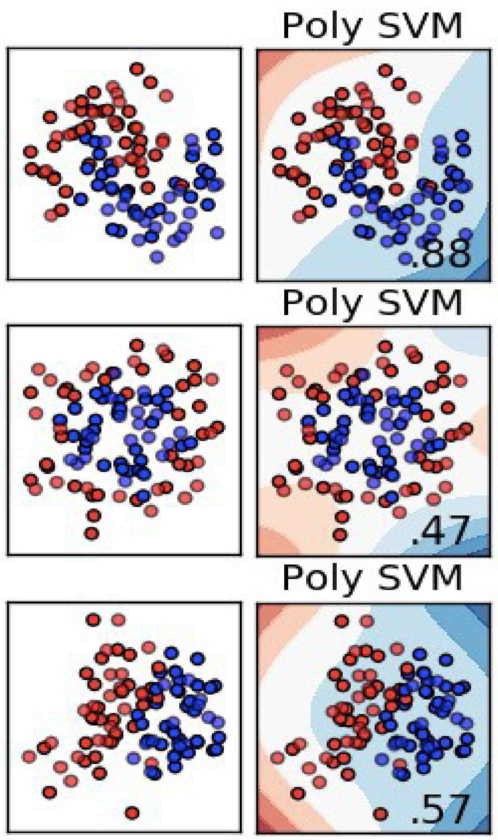
d =1: $\Phi(\vec{x})\Phi(\vec{z}) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = x_1 z_2 + x_1 z_2 = (\vec{x}\cdot\vec{z})$

Second degree polynomial mapping: $\Phi(\vec{x}) = \Phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{bmatrix}$

d =2: $\Phi(\vec{x})\Phi(\vec{z}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 \\ \sqrt{2}z_1 z_2 \\ z_2^2 \end{bmatrix} = x_1^2 z_1^2 + 2x_1 x_2\, z_1 z_2 + x_2^2 z_2^2$

$$= (x_1 z_1 + x_2 z_2)^2$$
$$= (\vec{x}\cdot\vec{z})^2$$

Example Decision Boundaries

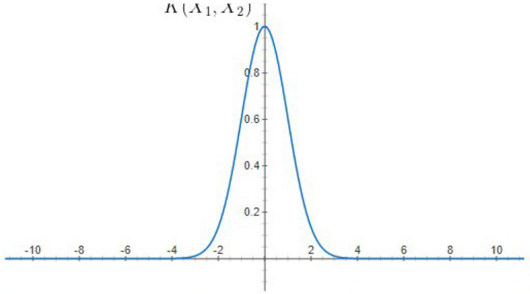


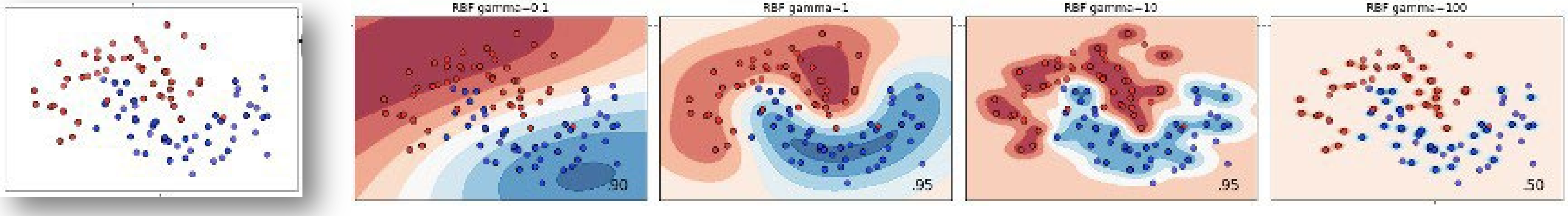
Kernel Function: RBF Kernel

- Radial Basis Function (RBF):

$$k(\vec{x},\vec{z}) = \exp\left(-\frac{\|\vec{x}-\vec{z}\|}{2\sigma^2}\right), \qquad k(\cdot) \in [0,1]$$

  - sigma (σ) defines kernel width (radius) of a single training data
    - i.e., high σ values mean 'wider kernel' → smoother decision boundary
    - i.e., small σ values mean 'narrow kernel' → like the nearest neighbors

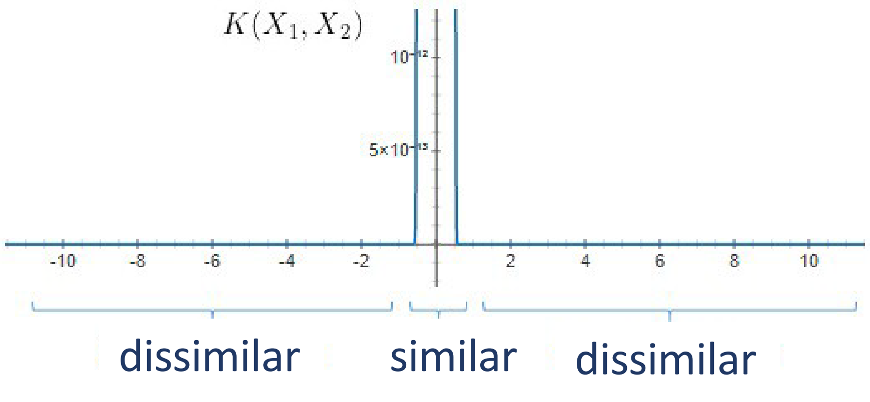




Example: Kernel Width (σ)

σ=0.1

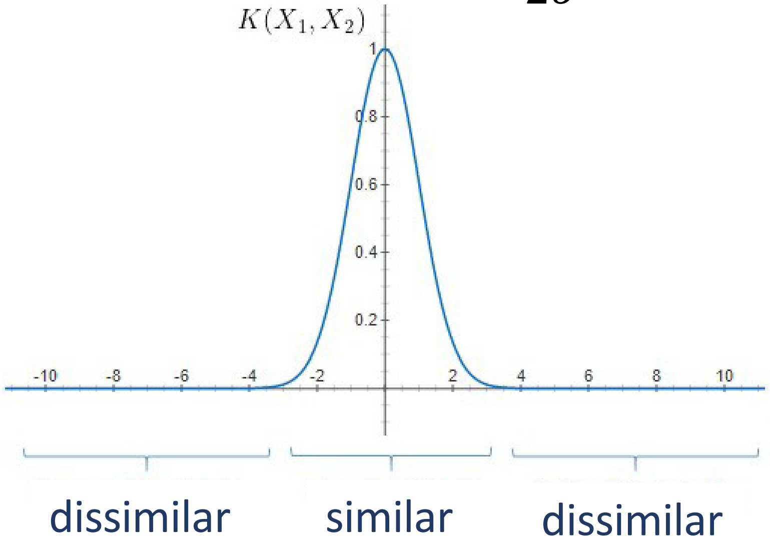
$$k(\vec{x},\vec{z}) = \exp(-\frac{\|\vec{x}-\vec{z}\|}{2\sigma^2})$$



(a) Narrower Kernel Width

σ=1

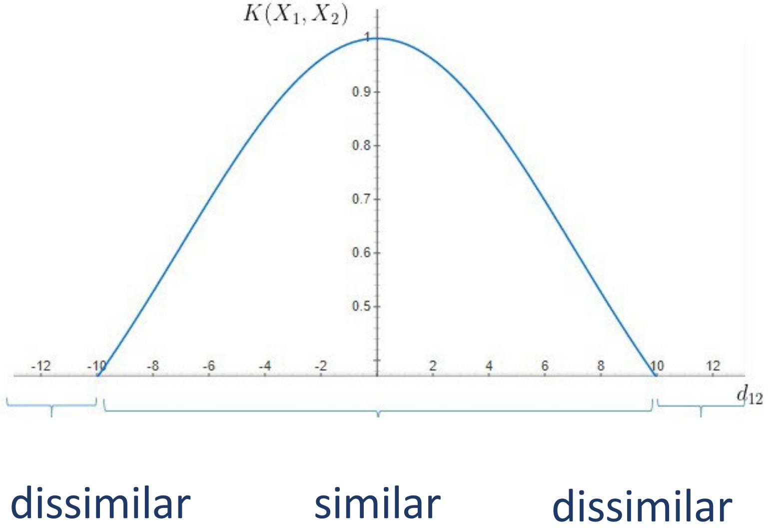
$$k(\vec{x},\vec{z}) = \exp(-\frac{\|\vec{x}-\vec{z}\|}{2\sigma^2})$$



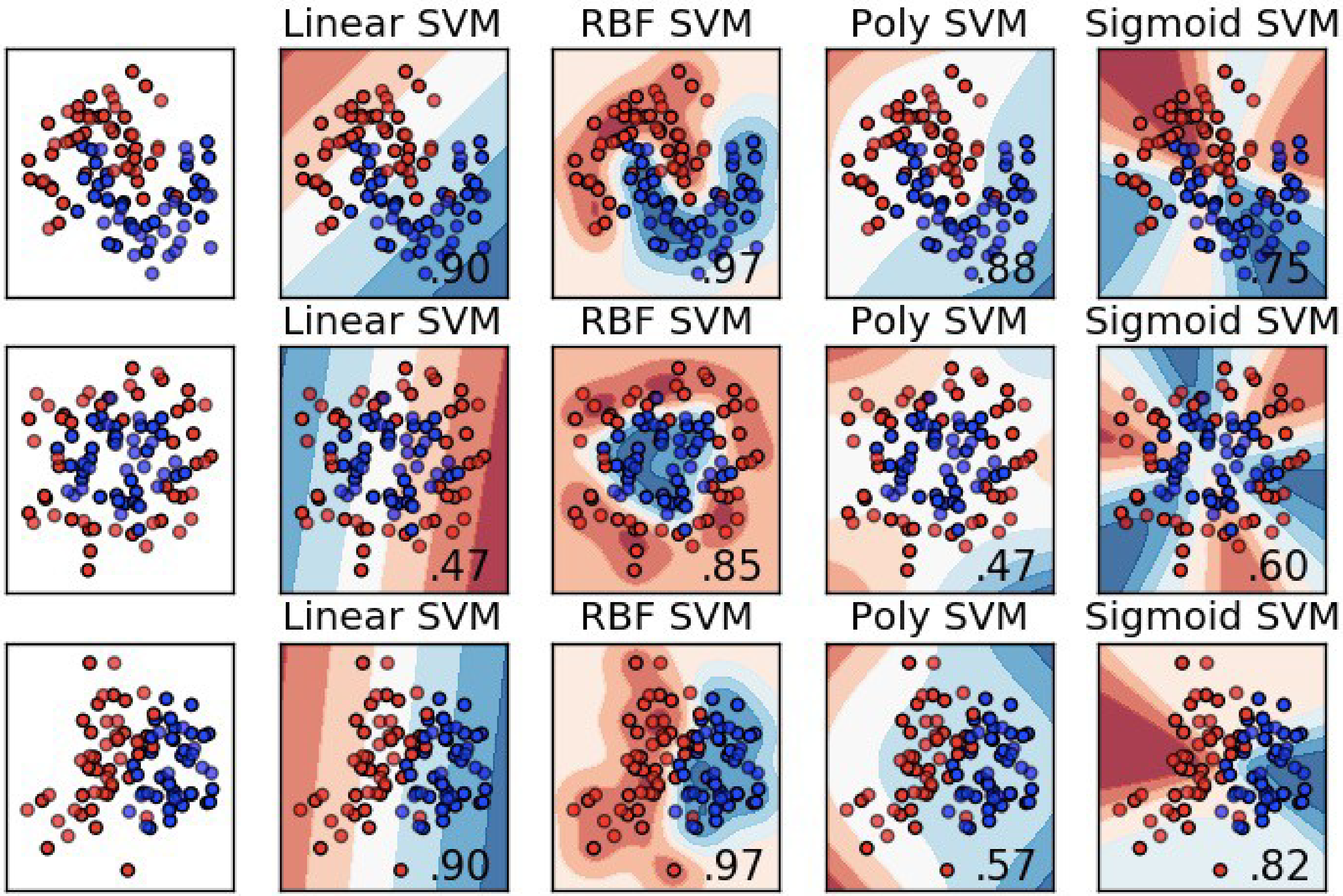
(b) Kernel Width

σ=10

$$k(\vec{x},\vec{z}) = \exp(-\frac{\|\vec{x}-\vec{z}\|}{2\sigma^2})$$



(c) Wider Kernel Width

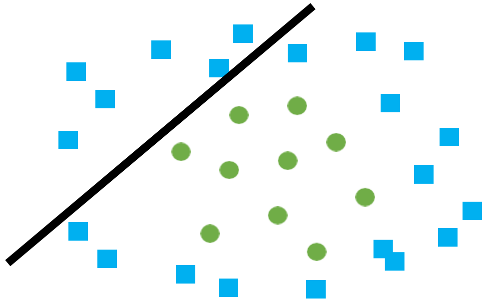


Source: https://www.kaggle.com/residentmario/kernels-and-support-vector-machine-regularization

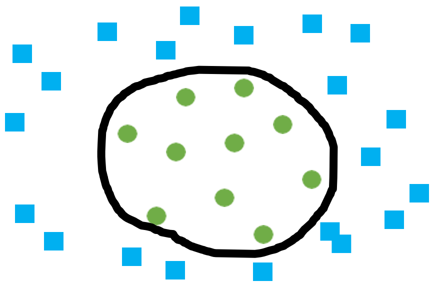
Bias and Variance

- A non-linear separation that trade-off between the bias and variance

High Bias

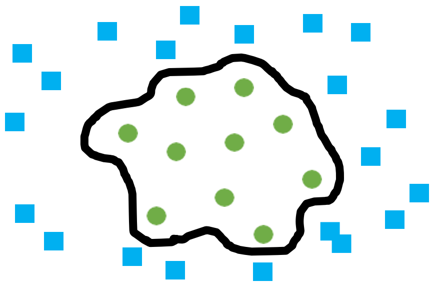


Underfitting



Good Compromise

Low Bias

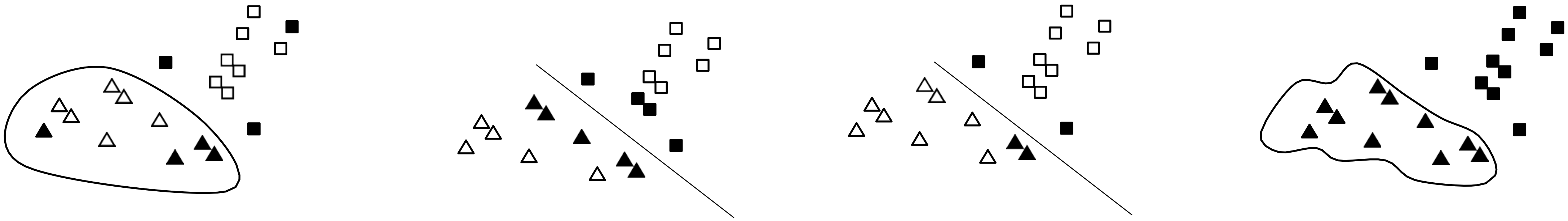


Overfitting

Quiz: Kernelized SVM with Soft/Hard Margin

- Which decision boundaries refer to kernelize SVMs? (Note: support vectors are represented by solid square/triangle)

Kernelized SVM **Hard-margin SVM** **Hard-margin SVM** **Kernelized SVM**



Advantages v.s. Disadvantages

Advantages

- Prediction accuracy is generally high
- As compared to Bayesian methods generally
- Robust when training examples contain errors
- Fast evaluation of the learned target function
- Bayesian networks are normally slow

Disadvantages

- Long training time
- Difficult to understand the learned function (weights)
- Bayesian networks can be used easily for pattern discovery
- Not easy to incorporate domain knowledge
- Easy in the form of priors on the data or distributions

Why Kernelized SVMs Work?

- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- The hyperplane is discovered based on support vectors (“essential” training tuples) and margins (defined by the support vectors)

Jupyter Notebook

Kernelized SVMs Coding Example

What about if we have more than 2 classes?

- Original SVMs can only perform binary classification (2 classes).

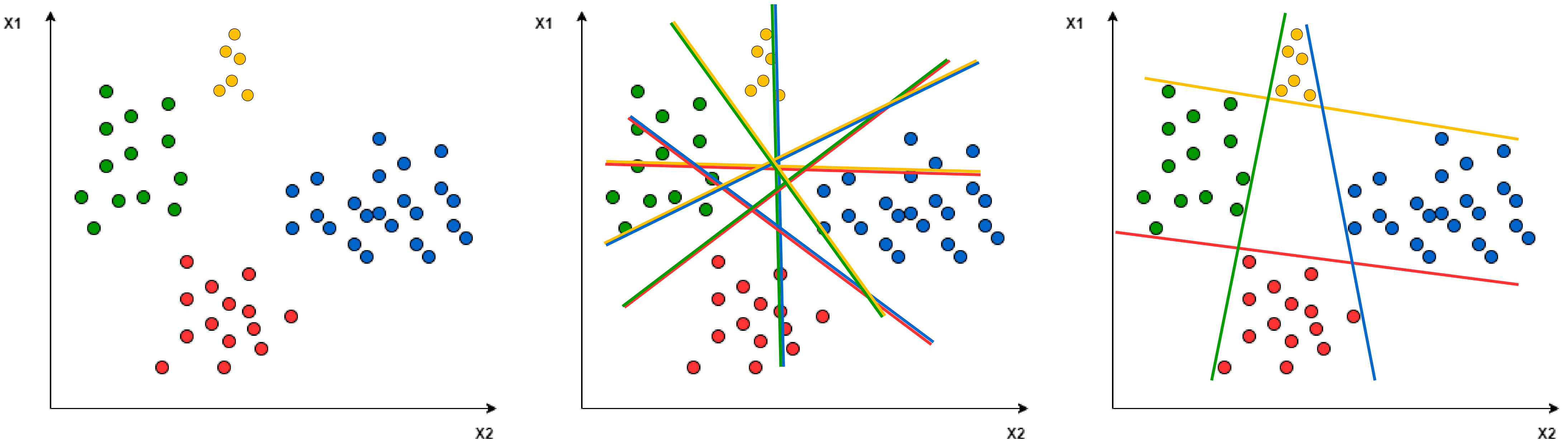
- SVM can be extended to multiclass problems (more then 2 classes).

- 2 approaches:
  1. One-vs-One (OVO):

     Train one SVMs for each binary problem, i.e., each 2 classes, ignoring the other ($\frac{m(m-1)}{2}$ SVMs in total).
  2. One-vs-Rest (OVR):

     Train *m* SVMs. Each SVM learns to separate 1 class from all the other ones.

OVO vs OVR



OVO

$\frac{m(m-1)}{2}$ = 6 SVMs

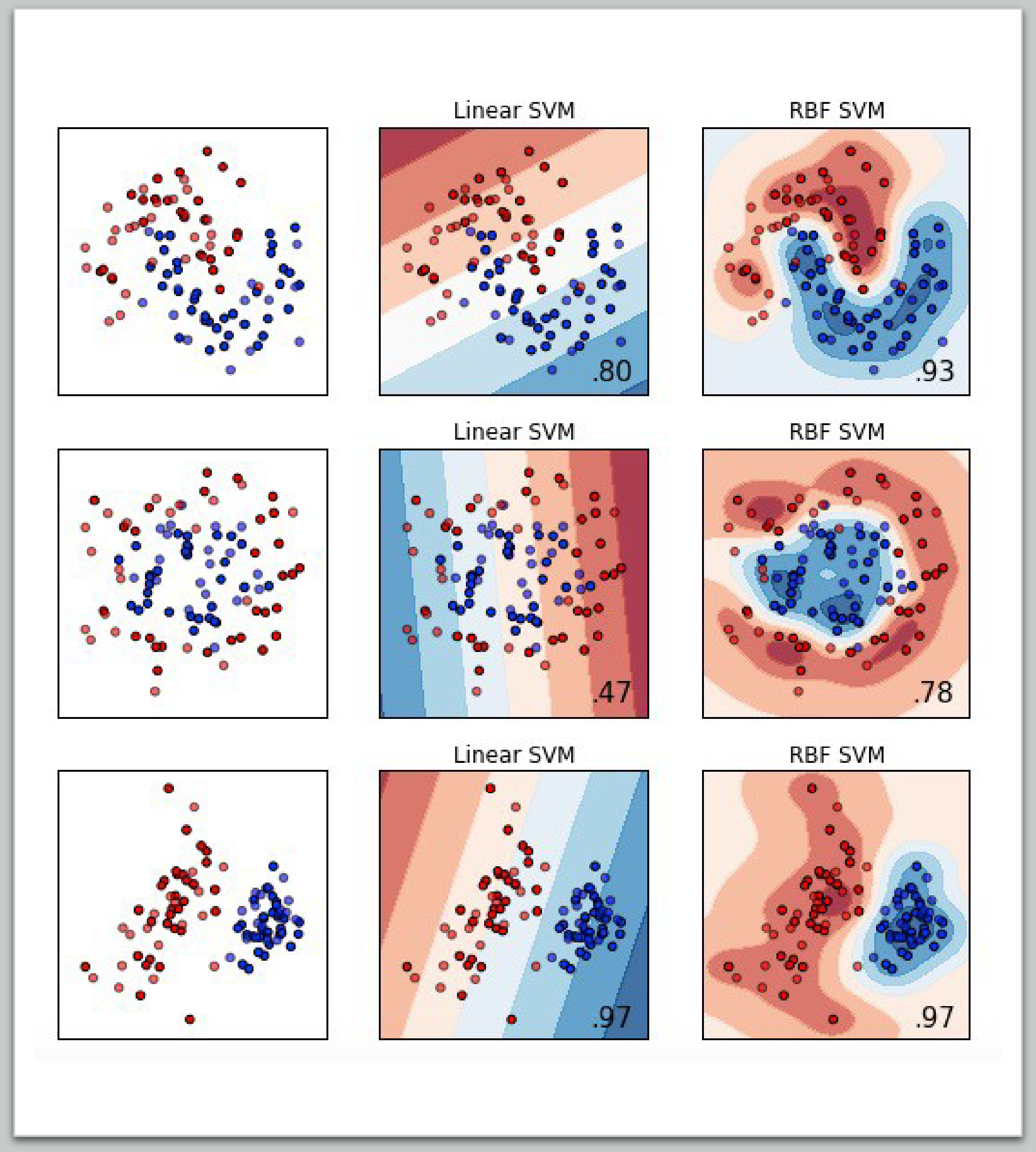
OVR

4 SVMs

SUMMARY

- Kernelized SVMs
  - Non-linearly Separable Data
  - Dual Problem Optimization
  - Kernel Problem Optimization
  - Testing Stage
  - Kernel Functions

- SVMs: Advantages and Disadvantages



Resources

- SVM Website: http://www.kernel-machines.org/

- Representative Implementation
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than LIBSVM, support only binary classification and only in C
  - **SVM-torch**: another recent implementation also written in C
  - **Scikit-Learn**: a set of supervised learning methods used for classification, regression and outliers detection. [link]

Resources (Contd.)

- Book Chapters: Christopher Bishop, “Pattern Recognition and Machine Learning” (PDF)
  - Sec 7.1.1-7.1.3
  - Sec 4.1.1, 4.1.2
  - Sec 6.1, 6.2
  - Appendix E

- Literatures
  - C.J.C. Burges, Chris J.C. Burges "A Tutorial on Support Vector Machines for Pattern Recognition.” Data Mining and Knowledge Discovery, 1998 (PDF)