

# **Kernel Methods**

## **CMSC 173 - Machine Learning**

---

Noel Jeffrey Pinton

October 3, 2025

Department of Computer Science  
University of the Philippines - Cebu

# Outline

---

Introduction & Motivation

Historical Context: From Perceptron to SVM

Support Vector Machines

Large-Margin Classifiers

Quadratic Optimization Problem

Nonlinear SVM using Kernels

Mercer's Theorem

Multiple Kernel Learning

Multi-class Classification

Kernel Methods for Regression

Support Vector Regression

Kernel Ridge Regression

Parametric vs Non-parametric Models

## Supervised Learning: Learning from Labeled Examples

### Supervised Learning

Learn a mapping or function:  $y = f(x)$

from inputs ( $x$ ) to outputs ( $y$ ),

*given a labelled set of input-output examples*

**Regression**

**Classification**

### Core Concept

Given labeled training data  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , learn a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  that maps inputs to outputs.

# Comparison of Supervised Learning Methods

Method	Model	Loss Function
Linear Regression	$y = w^T x + b$	$\sum_{i=1}^N (y_i - w^T x_i - b)^2$ Sum-of-squares loss
Logistic Regression	$P(y = 1 x) = \sigma(w^T x + b)$ $\sigma(z) = \frac{1}{1+e^{-z}}$	$-\sum_{i=1}^N y_i \log(\sigma(w^T x_i + b))$ $+(1 - y_i) \log(1 - \sigma(w^T x_i + b))$ Cross-entropy loss
Support Vector Machine	$y = \text{sign}(w^T x + b)$	$\frac{1}{2} \ w\ ^2 + C \sum_i \max(0, 1 - y_i(w^T x_i + b))$ Hinge loss + L2 regularization

## Today's Focus

We'll explore how **Support Vector Machines** use the **kernel trick** to solve non-linear problems while maintaining computational efficiency.

## **Introduction & Motivation**

---

# What are Kernel Methods?

## Definition

**Kernel methods** are a class of algorithms that use kernel functions to operate in high-dimensional feature spaces without explicitly computing the coordinates in that space.

## Key Idea:

- Transform data to higher dimensions where it becomes linearly separable
- Use **kernel trick** to avoid explicit transformation
- Compute inner products efficiently in feature space

## Why Kernel Methods?

Many real-world problems are **not linearly separable** in their original feature space.

## Common Applications:

- **Classification:** Support Vector Machines
- **Regression:** Support Vector Regression
- **Dimensionality Reduction:** Kernel PCA
- **Clustering:** Kernel K-means

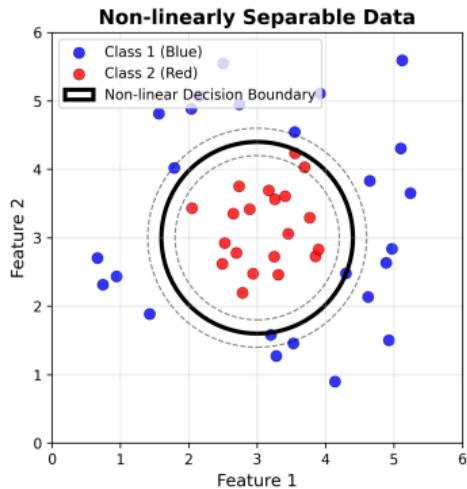
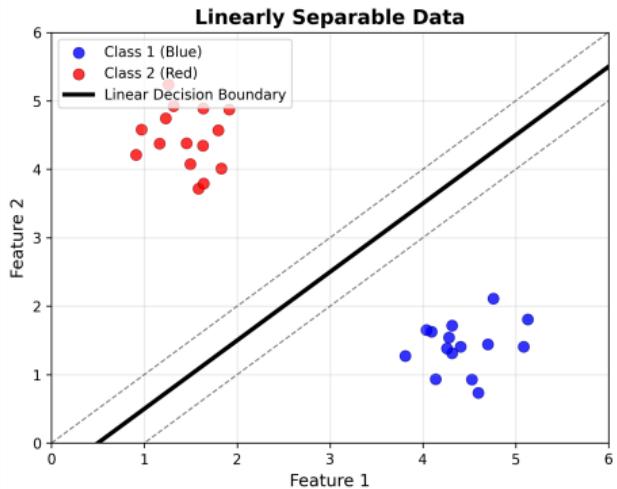
## Advantages:

- Handle non-linear relationships
- Computational efficiency via kernel trick
- Strong theoretical foundations
- Flexible and powerful

## Examples:

- Image classification
- Text analysis
- Bioinformatics
- Time series analysis

# Linear vs Non-linear Separability



## Linearly Separable Data

### Characteristics:

- Classes form distinct clusters
- Linear boundary separates perfectly
- Decision rule:  $w^T x + b = 0$

### Advantages:

- Simple and interpretable

## Non-linearly Separable Data

### Characteristics:

- Complex data patterns (e.g., concentric circles)
- No linear boundary can separate
- Requires non-linear decision boundary

### Solution: Kernel Methods

- Transform to higher dimensions

## Historical Context: From Perceptron to SVM

---

# Rosenblatt's Perceptron (1957)

## The Perceptron Algorithm

Frank Rosenblatt introduced the first learning algorithm for binary classification in 1957.

## Perceptron Model:

$$f(x) = \text{sign}(w^T x + b) \quad (1)$$

$$\text{Output} = \begin{cases} +1 & \text{if } w^T x + b \geq 0 \\ -1 & \text{if } w^T x + b < 0 \end{cases} \quad (2)$$

## Learning Rule:

$$w^{(t+1)} = w^{(t)} + \eta \cdot y_i \cdot x_i \quad (3)$$

$$b^{(t+1)} = b^{(t)} + \eta \cdot y_i \quad (4)$$

Update only when misclassified

## Perceptron Convergence Theorem

If data is linearly separable, the perceptron algorithm will converge to a solution in finite steps.

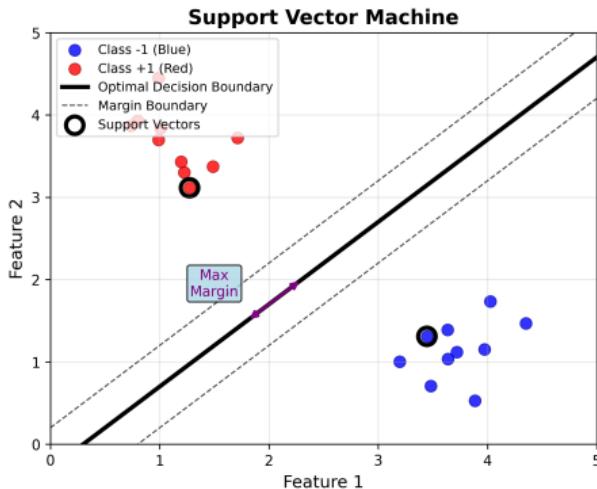
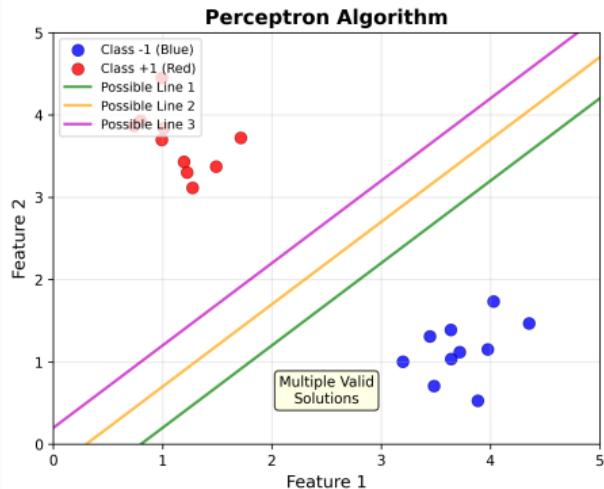
## Historical Impact:

- 1957: Perceptron introduced
- 1969: Limitations exposed (XOR problem)
- 1980s-1990s: SVM development
- Key insight: Maximum margin principle

## Motivation for SVMs:

- Address perceptron limitations
- Better generalization bounds
- Handle non-linearly separable data

# Perceptron vs SVM: A Visual Comparison



## Perceptron Approach

- Goal: Find any separating hyperplane
- Method: Iterative weight updates
- Result: Multiple valid solutions
- Issue: No optimality criterion

## SVM Approach

- Goal: Find optimal separating hyperplane
- Method: Maximize margin width
- Result: Unique optimal solution
- Benefit: Better generalization

## Key Insight

SVMs choose the hyperplane that maximizes the distance to the nearest data points (**support vectors**).

# From Linear to Non-linear: Evolution of Ideas

## Timeline of Development:

- 1957: Rosenblatt's Perceptron
- 1969: Minsky & Papert limitations
- 1979: Least squares SVM ideas
- 1992: Boser, Guyon, Vapnik kernel trick
- 1995: Cortes & Vapnik soft margin
- 1996: Schölkopf kernel methods

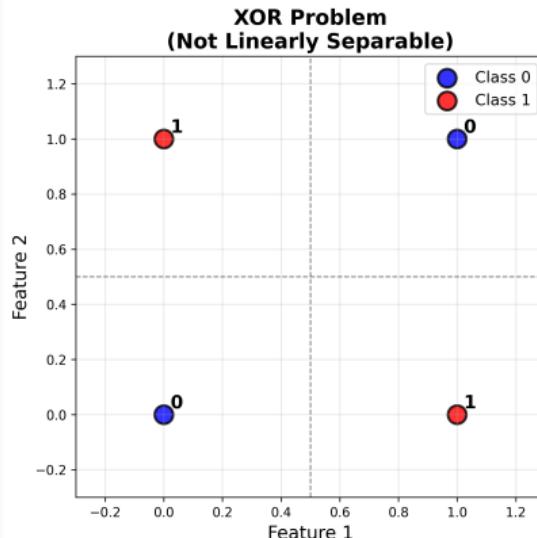
## Minsky & Papert (1969)

Showed that perceptrons cannot solve non-linearly separable problems like XOR, leading to the "AI winter."

## The Kernel Revolution

The kernel trick (1992) revived interest by enabling non-linear classification while maintaining computational efficiency.

## The XOR Problem:



## Problem Statement

- Input:  $(0, 0) \rightarrow 0, (0, 1) \rightarrow 1$
- Input:  $(1, 0) \rightarrow 1, (1, 1) \rightarrow 0$
- No linear separator exists

**Kernel Solution:** Transform to 3D space:

## **Support Vector Machines**

---

# Support Vector Machines: Overview

## Core Concept

SVM finds the optimal hyperplane that separates classes with the **maximum margin**.

## Key Components:

- **Decision boundary:** Hyperplane separating classes
- **Margin:** Distance between boundary and closest points
- **Support vectors:** Points defining the margin

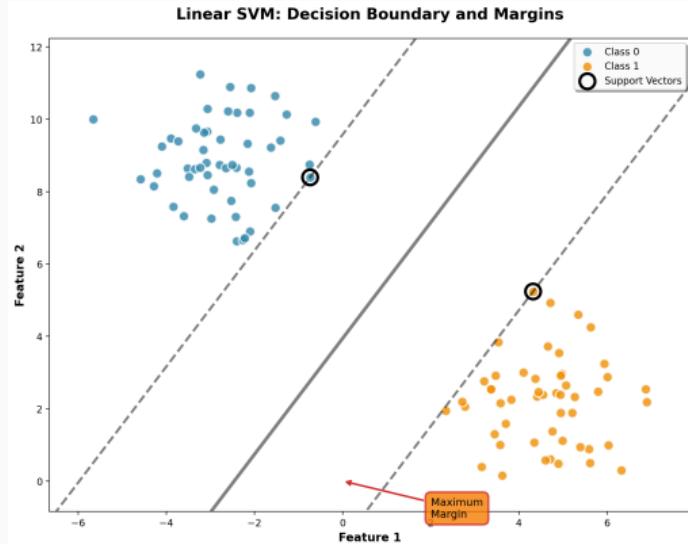
## Mathematical Formulation:

$$\text{Hyperplane: } w^T x + b = 0 \quad (5)$$

$$\text{Margin: } \frac{2}{\|w\|} \quad (6)$$

## Goal

**Maximize margin** while correctly classifying all training points.



## Why Maximum Margin?

- **Generalization:** Better performance on unseen data
- **Robustness:** Less sensitive to noise
- **Uniqueness:** Single optimal solution

## **Large-Margin Classifiers**

---

# Geometric Interpretation of Margin

## Margin Definition:

For a hyperplane  $w^T x + b = 0$ :

Distance from point  $x_i$  to hyperplane: (7)

$$d_i = \frac{|w^T x_i + b|}{\|w\|} \quad (8)$$

## Margin Width:

$$\text{Margin} = \min_i d_i = \frac{1}{\|w\|} \quad (9)$$

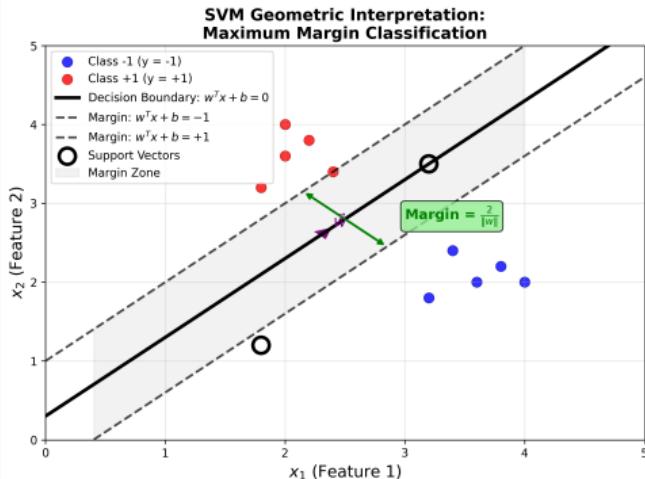
## Canonical Form

Scale  $w$  and  $b$  so that for the closest points:

$$w^T x_i + b = \pm 1$$

Then margin becomes  $\frac{2}{\|w\|}$ .

## Visual Interpretation:



## Key Elements

- **Decision boundary:**  $w^T x + b = 0$
- **Margin boundaries:**  $w^T x + b = \pm 1$
- **Support vectors:** Closest points
- **Margin width:**  $\frac{2}{\|w\|}$

## **Quadratic Optimization Problem**

---

## SVM Optimization: Primal Problem

Hard Margin SVM:

### Primal Problem

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad (10)$$

$$\text{subject to } y_i(w^T x_i + b) \geq 1, \quad i = 1, \dots, n \quad (11)$$

Soft Margin SVM:

### Primal Problem with Slack Variables

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad (12)$$

$$\text{subject to } y_i(w^T x_i + b) \geq 1 - \xi_i \quad (13)$$

$$\xi_i \geq 0, \quad i = 1, \dots, n \quad (14)$$

Interpretation:

- **Objective:** Minimize  $\|w\|^2 \Rightarrow$  Maximize margin  $\frac{2}{\|w\|}$
- **Constraints:** All points correctly classified with margin  $\geq 1$

Problem Type:

- Quadratic objective function
- Linear constraints
- Convex optimization problem
- Unique global optimum

Slack Variables  $\xi_i$ :

- $\xi_i = 0$ : Point correctly classified with margin  $\geq 1$
- $0 < \xi_i < 1$ : Point correctly classified but within margin
- $\xi_i \geq 1$ : Point misclassified

Regularization Parameter  $C$ :

- Large  $C$ : Penalty for violations (hard margin)
- Small  $C$ : Allow more violations (soft margin)
- Controls bias-variance tradeoff

## SVM Optimization: Dual Problem

### Lagrangian Formulation:

#### Lagrangian

$$L = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad (15)$$

$$- \sum_{i=1}^n \alpha_i [y_i(w^T x_i + b) - 1 + \xi_i] \quad (16)$$

$$- \sum_{i=1}^n \mu_i \xi_i \quad (17)$$

### Dual Problem:

#### Dual Formulation

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (21)$$

$$\text{subject to } \sum_{i=1}^n \alpha_i y_i = 0 \quad (22)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \quad (23)$$

### KKT Conditions:

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^n \alpha_i y_i x_i \quad (18)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^n \alpha_i y_i = 0 \quad (19)$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow \alpha_i + \mu_i = C \quad (20)$$

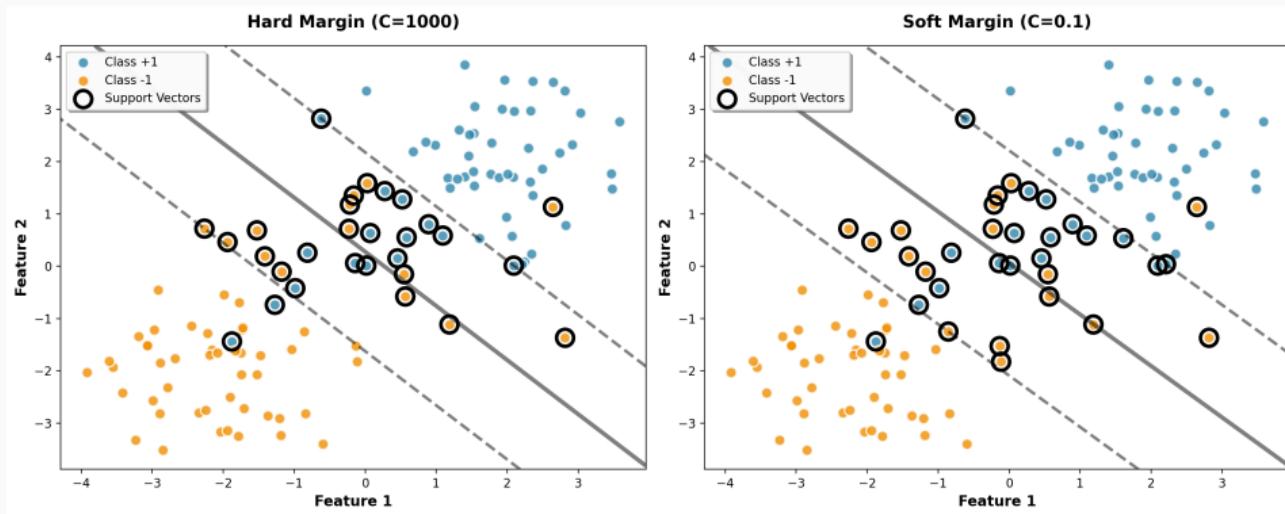
### Key Insights:

- Only depends on **inner products**  $x_i^T x_j$
- Sparse solution: many  $\alpha_i = 0$
- Support vectors:  $\alpha_i > 0$

### Kernel Trick Preview

Replace  $x_i^T x_j$  with  $K(x_i, x_j)$  to work in higher dimensions!

# Hard vs Soft Margin SVMs



## Hard Margin ( $C = 1000$ )

- No training errors allowed
- Requires linearly separable data
- May overfit to training data
- Complex decision boundary

## Soft Margin ( $C = 0.1$ )

- Allows some training errors
- Works with non-separable data
- Better generalization
- Smoother decision boundary

## **Nonlinear SVM using Kernels**

---

# The Kernel Trick

## Feature Mapping:

Transform input space  $\mathcal{X}$  to feature space  $\mathcal{H}$ :

$$\phi : \mathcal{X} \rightarrow \mathcal{H}$$

## Example: Polynomial Features

$$x = (x_1, x_2) \quad (24)$$

$$\phi(x) = (1, x_1, x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \quad (25)$$

## Kernel Function

Instead of computing  $\phi(x_i)^T \phi(x_j)$  explicitly:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

## Key Insight

We can compute inner products in high-dimensional space without explicitly mapping to that space!

## Computational Advantage:

### Without Kernel Trick:

- Map:  $\mathcal{O}(d')$  where  $d'$  is feature space dimension
- Inner product:  $\mathcal{O}(d')$
- Total:  $\mathcal{O}(d')$  per pair

### With Kernel Trick:

- Direct kernel computation:  $\mathcal{O}(d)$  where  $d$  is input dimension
- No explicit mapping needed
- Total:  $\mathcal{O}(d)$  per pair

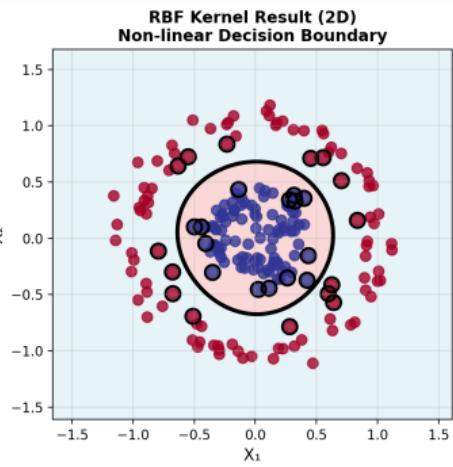
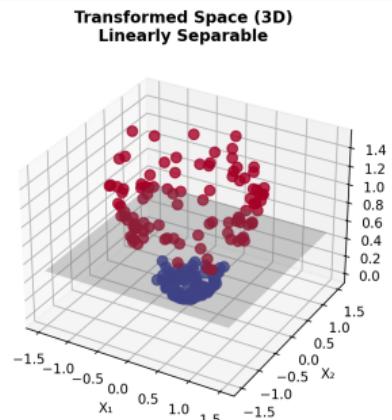
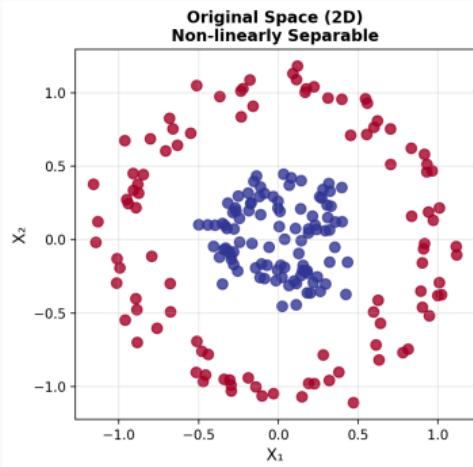
## Example: Polynomial Kernel

$$K(x, x') = (x^T x' + 1)^2 \quad (26)$$

$$= 1 + 2x^T x' + (x^T x')^2 \quad (27)$$

This implicitly computes inner product in a  $(d+1)(d+2)/2$  dimensional space using only  $\mathcal{O}(d)$  operations!

# Kernel Trick Visualization



## Original Space

Non-linearly separable data in 2D  
cannot be separated by a straight  
line.

## Transformed Space

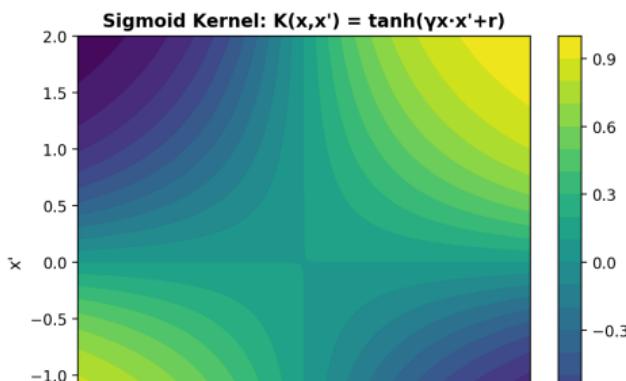
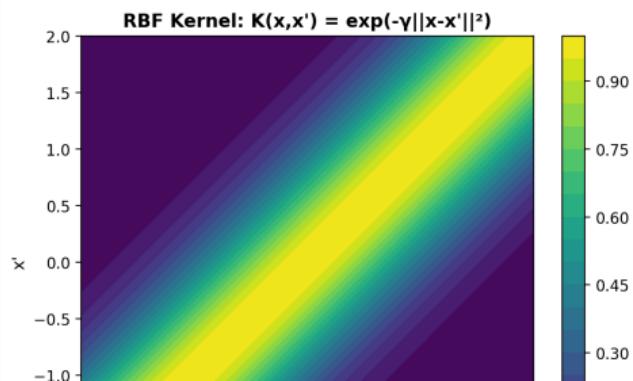
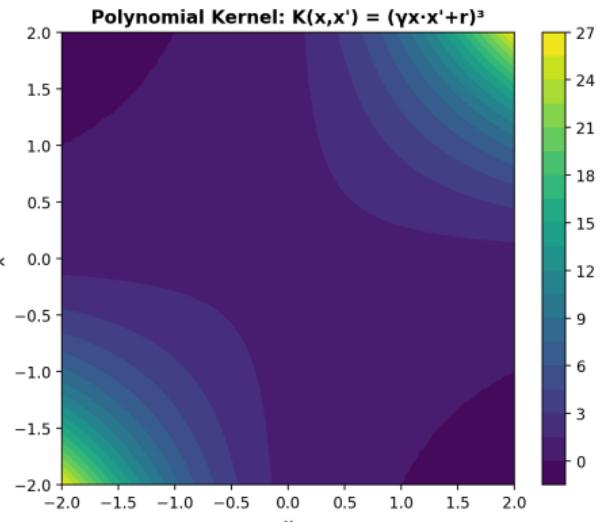
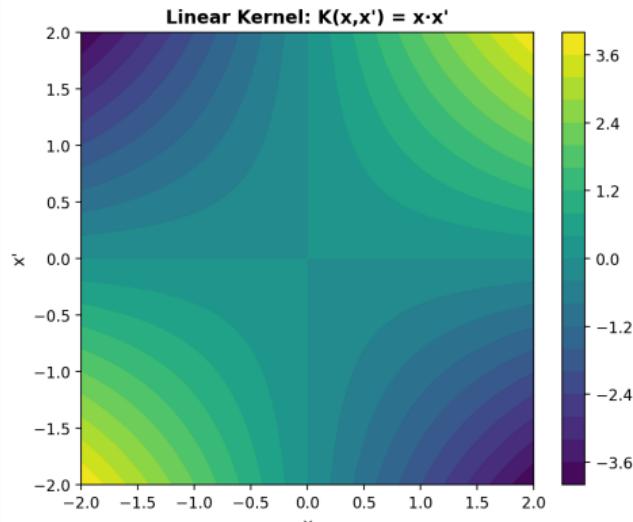
Data becomes linearly separable in  
3D after polynomial  
transformation

$$\phi(x) = (x_1, x_2, x_1^2 + x_2^2).$$

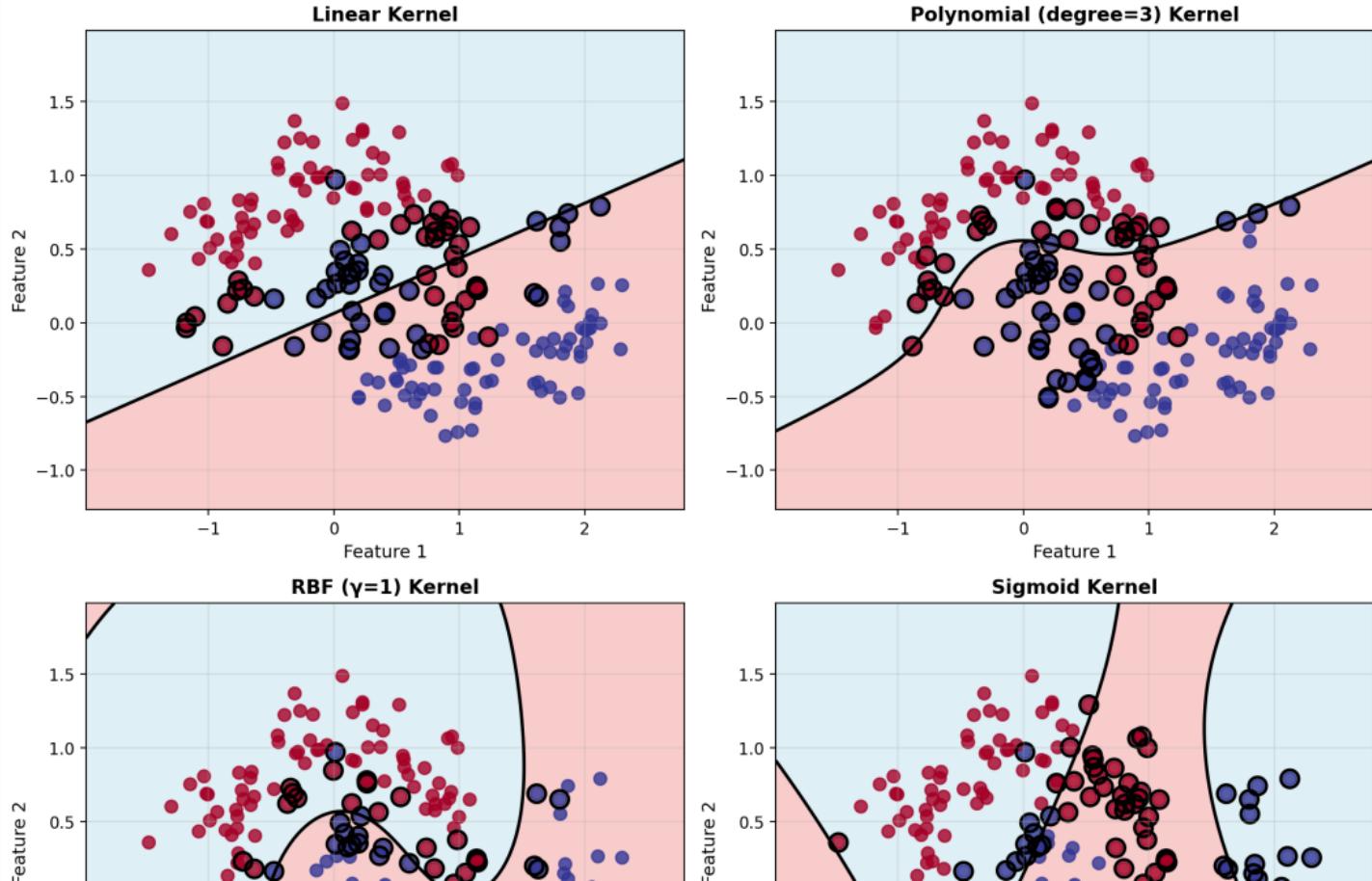
## Kernel Result

RBF kernel achieves non-linear  
separation directly in original 2D  
space.

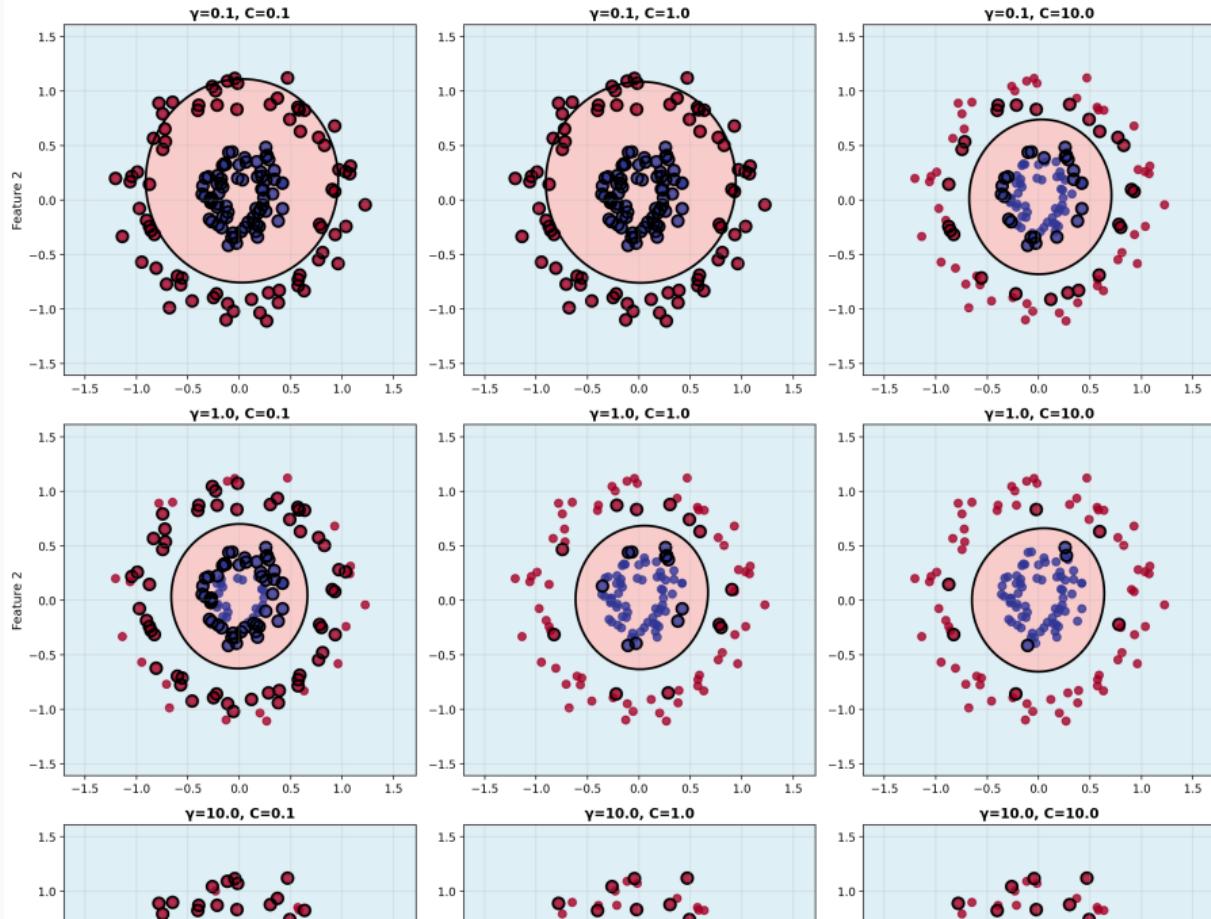
## Common Kernel Functions



## Comparing Different Kernels



## RBF Kernel Parameter Effects



## **Mercer's Theorem**

---

## Mercer's Theorem: Valid Kernels

### Mercer's Theorem

A function  $K(x, x')$  is a valid kernel (corresponds to an inner product in some feature space) if and only if for any finite set of points  $\{x_1, \dots, x_n\}$ , the kernel matrix  $\mathbf{K}$  is positive semi-definite.

### Kernel Matrix:

$$\mathbf{K}_{ij} = K(x_i, x_j)$$

### Positive Semi-definite:

$$\mathbf{K} \succeq 0 \iff \sum_{i,j} c_i c_j K(x_i, x_j) \geq 0$$

for all real numbers  $c_1, \dots, c_n$ .

### Practical Implication

Not every function can be used as a kernel! Only those satisfying Mercer's condition.

### Properties of Valid Kernels:

- **Symmetry:**  $K(x, x') = K(x', x)$
- **Positive semi-definiteness:** Kernel matrix  $\mathbf{K} \succeq 0$

### Constructing New Kernels:

If  $K_1$  and  $K_2$  are valid kernels, then:

$$K(x, x') = K_1(x, x') + K_2(x, x') \quad (28)$$

$$K(x, x') = c \cdot K_1(x, x'), \quad c > 0 \quad (29)$$

$$K(x, x') = K_1(x, x') \cdot K_2(x, x') \quad (30)$$

$$K(x, x') = \exp(K_1(x, x')) \quad (31)$$

$$K(x, x') = f(x)K_1(x, x')f(x') \quad (32)$$

### Domain-Specific Kernels:

- String kernels for text
- Graph kernels for networks
- Tree kernels for structured data

## Examples of Valid and Invalid Kernels

### Valid Kernels:

#### Linear

$$K(x, x') = x^T x'$$

#### Polynomial

$$K(x, x') = (x^T x' + 1)^d, \quad d \geq 1$$

#### RBF/Gaussian

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

#### Exponential

$$K(x, x') = \exp(-\gamma \|x - x'\|), \quad \gamma > 0$$

**Why Valid?** All can be shown to correspond to inner products in (possibly infinite-dimensional) feature spaces.

### Invalid Kernels:

#### Negative Power

$$K(x, x') = (x^T x')^{-1}$$

Not positive semi-definite.

#### Logarithmic

$$K(x, x') = \log(x^T x' + 1)$$

Can produce negative eigenvalues.

### Checking Validity:

1. **Theoretical:** Prove positive semi-definiteness
2. **Computational:** Check eigenvalues of kernel matrix
3. **Construction:** Build from known valid kernels

#### Practical Note

Most standard kernels in ML libraries are guaranteed to be valid. Custom kernels need verification.

## Multiple Kernel Learning

---

# Multiple Kernel Learning (MKL)

## Motivation

Different kernels capture different aspects of data:

- RBF kernel: local similarities
- Linear kernel: global structure
- Polynomial kernel: feature interactions

**Idea:** Combine multiple kernels to get better performance than any single kernel.

## Linear Combination:

$$K(x, x') = \sum_{m=1}^M \beta_m K_m(x, x')$$

where  $\beta_m \geq 0$  and  $\sum_{m=1}^M \beta_m = 1$ .

## Applications:

- Multi-modal data (text + images)
- Feature selection
- Domain adaptation

## MKL Optimization:

### Joint Optimization

$$\min_{w, b, \xi, \beta} \frac{1}{2} \sum_{m=1}^M \frac{\|w_m\|^2}{\beta_m} + C \sum_{i=1}^n \xi_i \quad (33)$$

$$\text{subject to } y_i \left( \sum_{m=1}^M w_m^T \phi_m(x_i) + b \right) \geq 1 - \xi_i \quad (34)$$

$$\xi_i \geq 0, \quad \beta_m \geq 0, \quad \sum_{m=1}^M \beta_m = 1 \quad (35)$$

## Solution Methods:

- **Alternating optimization:** Fix  $\beta$ , solve for  $w$ ; fix  $w$ , solve for  $\beta$
- **Semi-definite programming:** Convex formulation
- **Gradient-based methods:** Efficient for large-scale

## Kernel Weight Interpretation:

- $\beta_m$  close to 1: Kernel  $m$  is most important

## MKL Example: Combining Kernels

### Example Setup:

Consider combining three kernels:

$$K_1(x, x') = x^T x' \quad (\text{Linear}) \quad (36)$$

$$K_2(x, x') = (x^T x' + 1)^2 \quad (\text{Polynomial}) \quad (37)$$

$$K_3(x, x') = \exp(-\|x - x'\|^2) \quad (\text{RBF}) \quad (38)$$

### Sample Results:

Kernel	Weight	Accuracy
Linear	0.1	0.78
Polynomial	0.3	0.82
RBF	0.6	0.85
<b>Combined MKL</b>	-	<b>0.89</b>

### Combined Kernel:

$$K(x, x') = \beta_1 K_1(x, x') + \beta_2 K_2(x, x') + \beta_3 K_3(x, x')$$

### Learning Process:

1. Start with equal weights:  $\beta_1 = \beta_2 = \beta_3 = \frac{1}{3}$
2. Iteratively optimize weights and SVM parameters
3. Converge to optimal combination

### Advantages:

- Better performance than individual kernels
- Automatic selection of relevant kernels
- Interpretability through weights

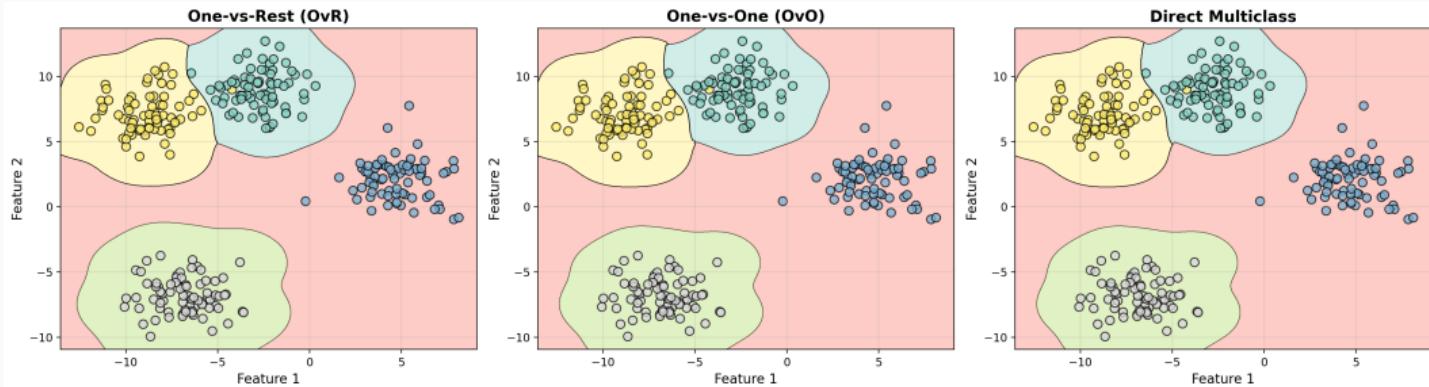
### Challenges:

- Increased computational complexity
- More hyperparameters to tune
- Risk of overfitting with many kernels

## **Multi-class Classification**

---

# Multi-class SVM Strategies



## One-vs-Rest (OvR)

- Train  $k$  binary classifiers
- Each separates one class from all others
- Prediction: class with highest score

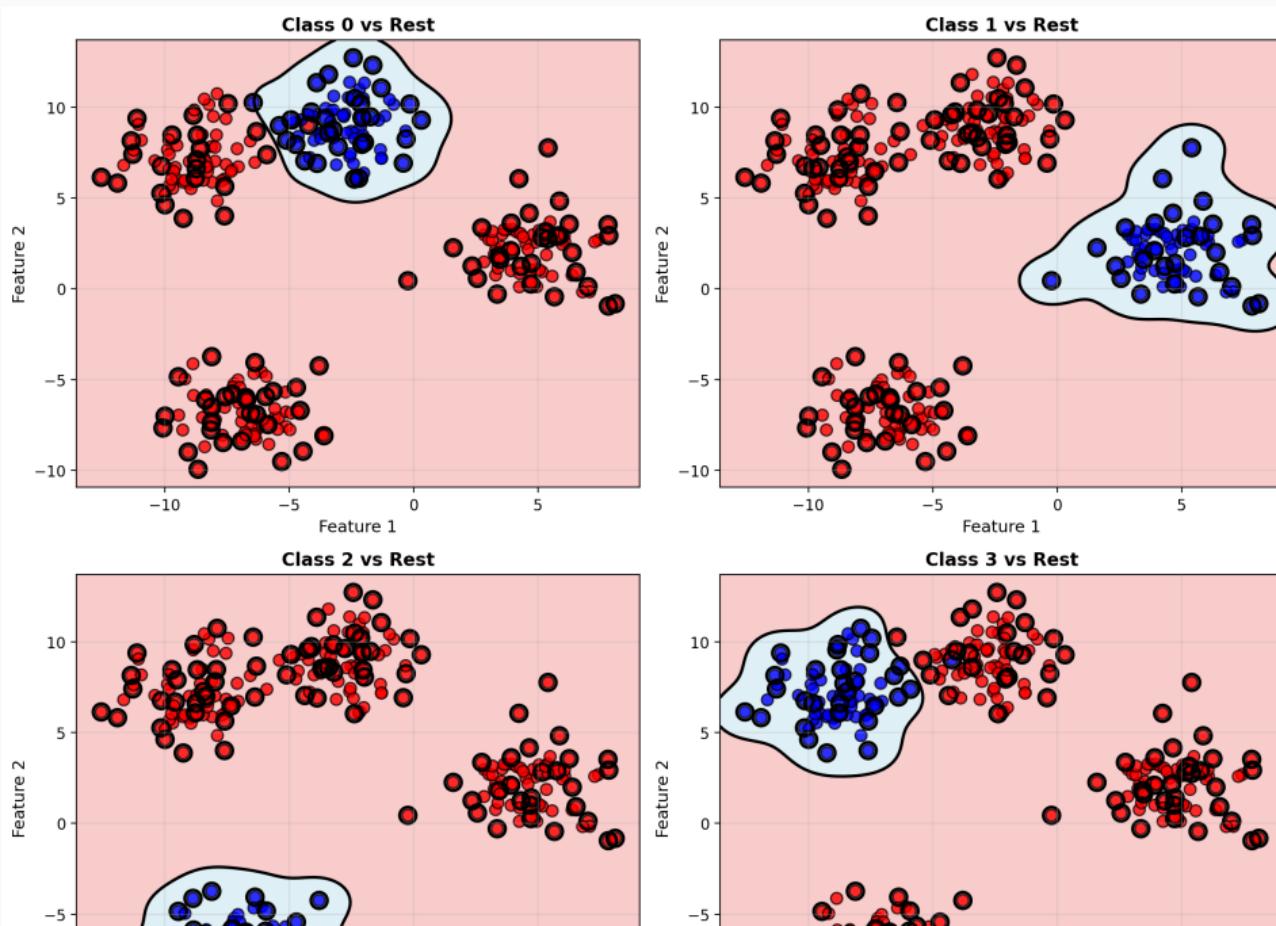
## One-vs-One (OvO)

- Train  $\binom{k}{2}$  binary classifiers
- Each separates pair of classes
- Prediction: majority voting

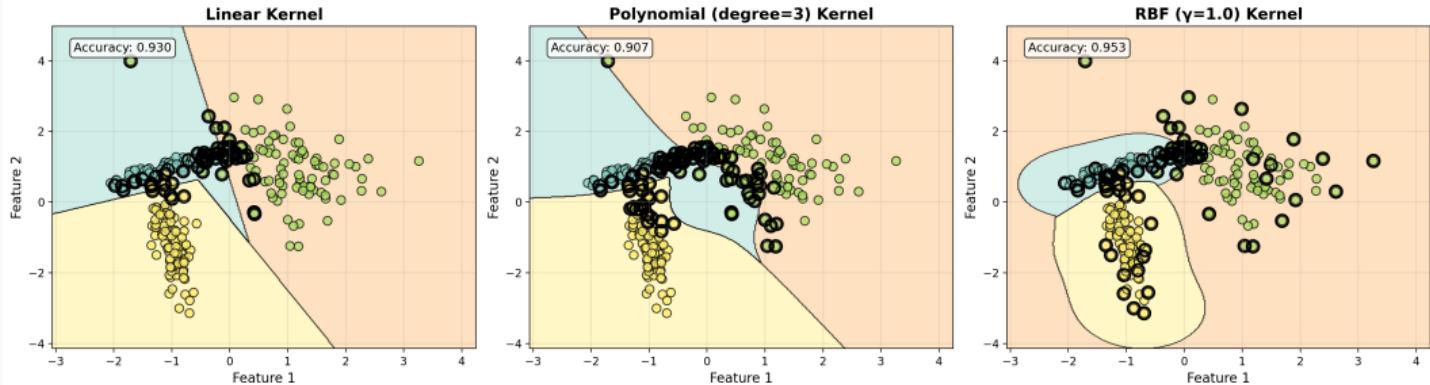
## Direct Multiclass

- Single optimization problem
- Simultaneous separation
- More complex but unified

## One-vs-Rest Detailed Analysis



# Multi-class Kernel Comparison



## Performance Analysis:

- **Linear Kernel:** Simple boundaries, good for high-dim data
- **Polynomial Kernel:** Captures feature interactions
- **RBF Kernel:** Most flexible, handles complex patterns

## Selection Criteria:

- Dataset size and dimensionality
- Computational resources

## Multi-class Considerations:

- **Class balance:** Equal vs imbalanced classes
- **Separability:** Linear vs non-linear boundaries
- **Noise sensitivity:** Robust vs sensitive kernels

## Hyperparameter Tuning:

- Grid search over kernel parameters
- Cross-validation for each class combination
- Balanced accuracy metrics

## Kernel Methods for Regression

---

# Support Vector Regression (SVR)

## Key Parameters:

- $\epsilon$ : Width of insensitive zone
- $C$ : Regularization parameter
- **Kernel parameters:**  $\gamma$  for RBF, etc.

## SVR Concept

Extend SVM to regression by finding a function that deviates from target values by at most  $\epsilon$ , while being as flat as possible.

## Linear SVR:

$$f(x) = w^T x + b$$

## Optimization Problem:

$$\min_{w, b, \xi, \xi^*} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*) \quad (39)$$

$$\text{subject to } y_i - w^T x_i - b \leq \epsilon + \xi_i \quad (40)$$

$$w^T x_i + b - y_i \leq \epsilon + \xi_i^* \quad (41)$$

$$\xi_i, \xi_i^* \geq 0 \quad (42)$$

## $\epsilon$ -insensitive Loss:

$$L_\epsilon(y, f(x)) = \max(0, |y - f(x)| - \epsilon)$$

## Dual Formulation:

$$f(x) = \sum_{i=1}^n (\alpha_i - \alpha_i^*) K(x_i, x) + b \quad (43)$$

where  $\alpha_i, \alpha_i^* \geq 0$  are Lagrange multipliers.

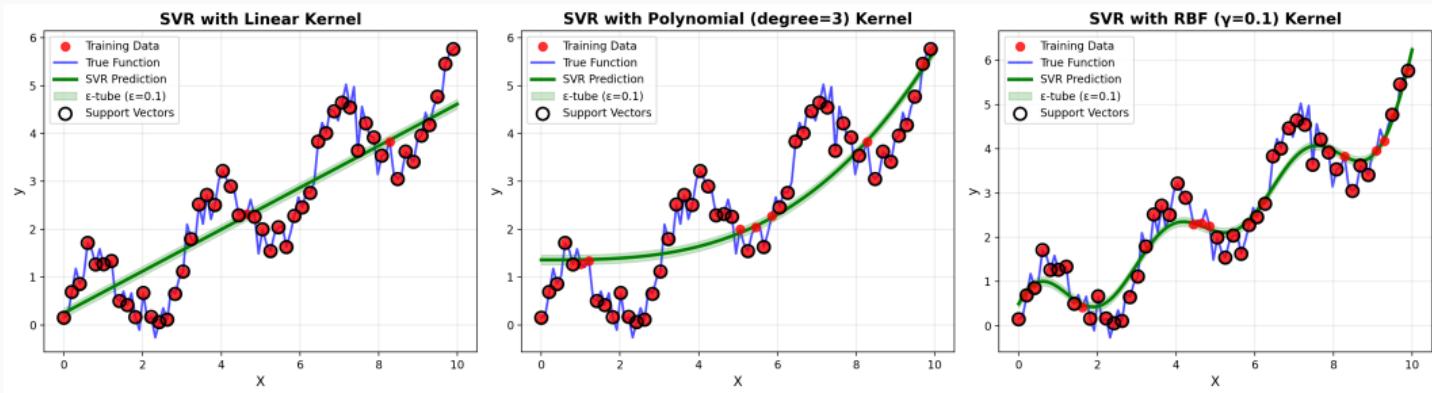
## Support Vectors:

- Points outside  $\epsilon$ -tube
- $\alpha_i > 0$  or  $\alpha_i^* > 0$
- Determine the regression function

## Sparsity

Many  $\alpha_i = \alpha_i^* = 0$ , leading to sparse solutions.

# SVR Demonstration



## Linear SVR

- Simple linear relationship
- Good for linear trends
- Fast computation

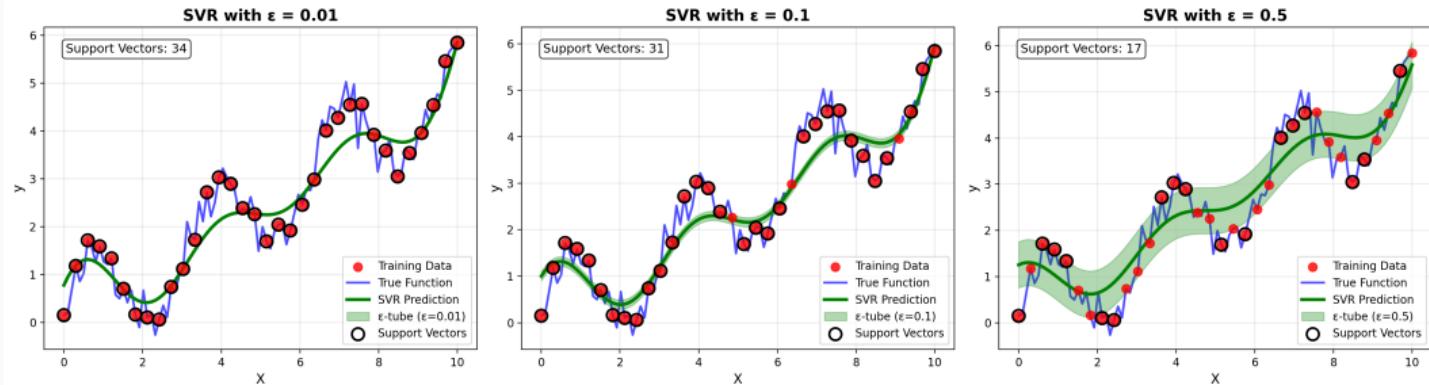
## Polynomial SVR

- Captures polynomial trends
- Risk of overfitting
- Degree selection important

## RBF SVR

- Most flexible
- Handles non-linear patterns
- Requires parameter tuning

# Effect of $\epsilon$ Parameter in SVR



## $\epsilon$ Parameter Effects:

- **Small  $\epsilon$  (0.01):** Tight fit, many support vectors
- **Medium  $\epsilon$  (0.1):** Balanced complexity
- **Large  $\epsilon$  (0.5):** Loose fit, fewer support vectors

## Trade-offs:

- **Small  $\epsilon$ :** Low bias, high variance
- **Large  $\epsilon$ :** High bias, low variance
- **Sparsity:** Larger  $\epsilon \Rightarrow$  fewer support vectors

## Selection Guidelines:

- Cross-validation for optimal  $\epsilon$
- Consider noise level in data
- Balance accuracy vs complexity

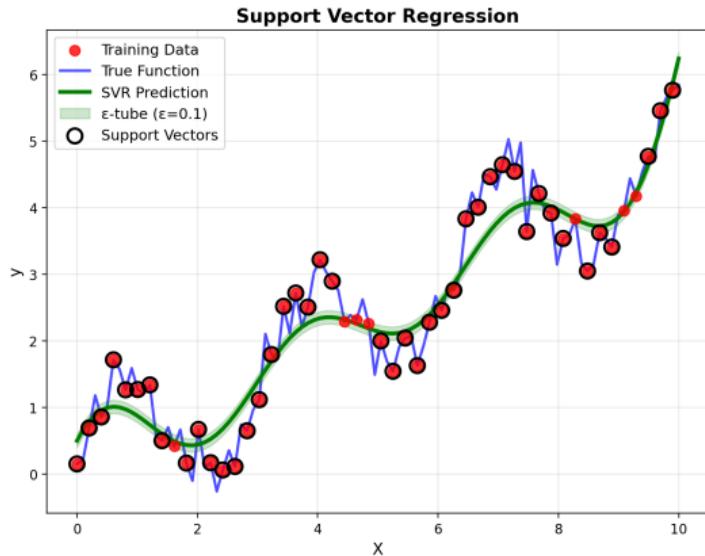
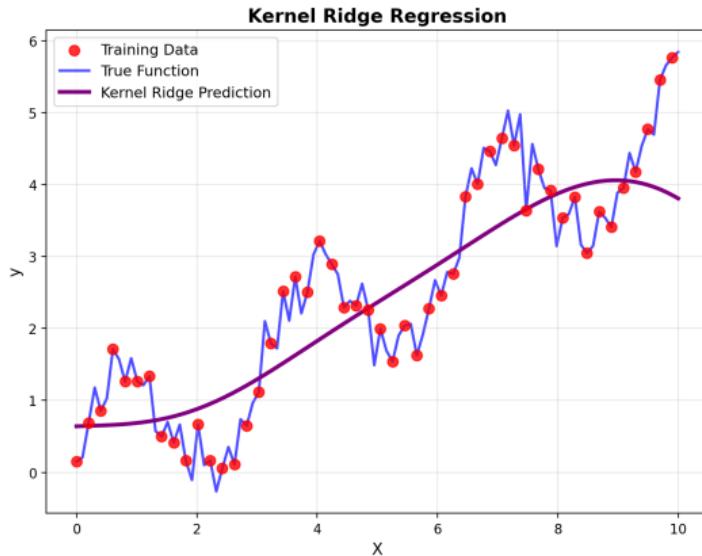
## Practical Values:

- Start with  $\epsilon = 0.1$
- Scale with target variable range
- Grid search with  $C$  and kernel parameters

## Support Vector Regression

---

# Kernel Ridge Regression vs SVR



## Kernel Ridge Regression

Objective:

$$\min_{\alpha} \|\mathbf{K}\alpha - y\|^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Solution:

$$\alpha = (\mathbf{K} + \lambda \mathbf{I})^{-1} y$$

## Support Vector Regression

Objective:

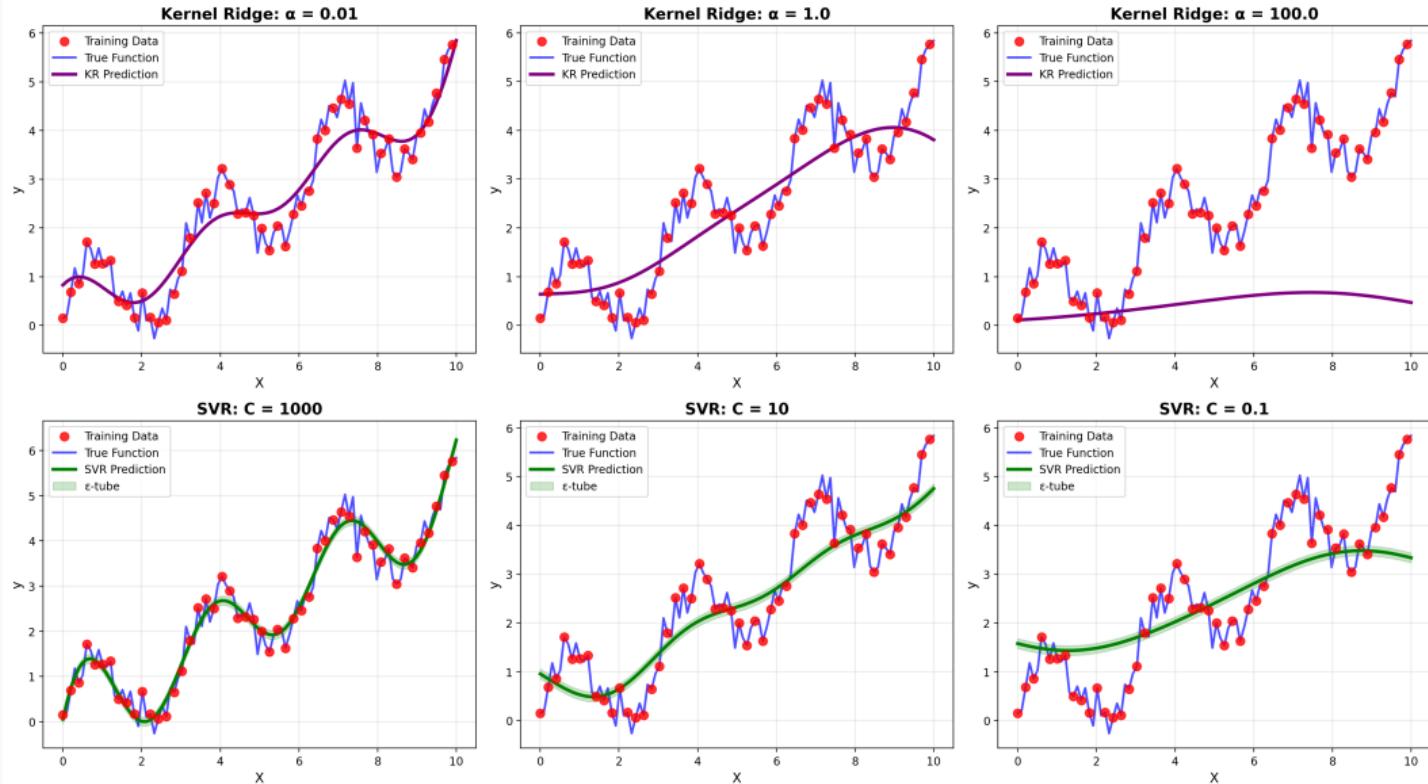
$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_i (\xi_i + \xi_i^*)$$

Constraints:

## Kernel Ridge Regression

---

# Regularization in Kernel Regression



Kernel Ridge Regularization ( $\alpha$ ):

- Small  $\alpha$  (0.01): Low regularization, overfitting

SVR Regularization ( $C$ ):

- High  $C$  (1000): Low regularization, complex

## Worked Example: RBF Kernel Computation

### Problem Setup:

Given two points:

$$x_1 = (1, 2) \quad (46)$$

$$x_2 = (3, 1) \quad (47)$$

Compute RBF kernel with  $\gamma = 0.5$ :

$$K(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$$

### Step 1: Compute distance

$$x_1 - x_2 = (1, 2) - (3, 1) = (-2, 1) \quad (48)$$

$$\|x_1 - x_2\|^2 = (-2)^2 + 1^2 = 4 + 1 = 5 \quad (49)$$

### Step 2: Apply kernel

$$K(x_1, x_2) = \exp(-0.5 \times 5) \quad (50)$$

$$= \exp(-2.5) \quad (51)$$

$$\approx 0.082 \quad (52)$$

### Interpretation:

- Points are moderately far apart
- Kernel value is small (0.082)
- Indicates low similarity

### Compare with closer points:

For  $x_1 = (1, 2)$  and  $x_3 = (1.1, 2.1)$ :

$$\|x_1 - x_3\|^2 = (0.1)^2 + (0.1)^2 = 0.02 \quad (53)$$

$$K(x_1, x_3) = \exp(-0.5 \times 0.02) = \exp(-0.01) \quad (54)$$

$$\approx 0.99 \quad (55)$$

### Effect of $\gamma$ :

- Large  $\gamma$ : Rapid decay, local influence
- Small  $\gamma$ : Slow decay, global influence

### Key Insight

RBF kernel measures similarity through Euclidean distance in input space.

# Practical Implementation Tips

## Data Preprocessing:

- **Feature Scaling:** Critical for RBF kernels
- **Normalization:** StandardScaler or MinMaxScaler
- **Missing Values:** Handle before kernel computation

## Model Selection:

1. Start with RBF kernel
2. Use cross-validation
3. Compare with linear kernel
4. Consider computational constraints

## Hyperparameter Tuning:

### Grid Search Example

```
param_grid = {  
    'C': [0.1, 1, 10, 100],  
    'gamma': [0.001, 0.01, 0.1, 1],  
    'kernel': ['rbf', 'poly', 'linear']  
}
```

## Performance Considerations:

- Linear kernel:  $\mathcal{O}(n \times d)$
- RBF kernel:  $\mathcal{O}(n \times d)$  per evaluation
- Training complexity:  $\mathcal{O}(n^2)$  to  $\mathcal{O}(n^3)$

## Common Pitfalls:

### Avoid These

- Forgetting to scale features
- Using default parameters
- Ignoring class imbalance
- Overfitting with complex kernels

## Debugging Tips:

- Check kernel matrix properties
- Visualize decision boundaries
- Monitor support vector counts
- Validate on holdout set

## Software Libraries:

## **Parametric vs Non-parametric Models**

---

# Understanding Model Types

## Parametric Models:

### Definition

Fixed number of parameters independent of training set size. Make strong assumptions about functional form.

### Examples:

- **Linear Regression:**  $f(x) = w^T x + b$
- **Logistic Regression:**  $p = \sigma(w^T x + b)$
- **Perceptron:** Fixed decision boundary
- **Neural Networks:** Fixed architecture

### Characteristics:

- Fast training and prediction
- Strong inductive bias
- May underfit complex data
- Interpretable parameters

## Non-parametric Models:

### Definition

Number of parameters grows with training data size. Make minimal assumptions about functional form.

### Examples:

- **k-NN:** Stores all training data
- **Decision Trees:** Adaptive structure
- **Kernel Methods:** Support vector representation
- **Gaussian Processes:** Infinite parameters

### Characteristics:

- Flexible representation
- Can fit complex patterns
- Risk of overfitting
- Higher computational cost

## Why SVMs are Non-parametric:

### Key Insight

SVM decision function depends on support vectors, whose number grows with data complexity, not fixed in advance.

## Decision Function:

$$f(x) = \sum_{i \in SV} \alpha_i y_i K(x_i, x) + b$$

- Number of support vectors  $|SV|$  varies
- Complex data  $\Rightarrow$  more support vectors
- Simple data  $\Rightarrow$  fewer support vectors

## Adaptive Complexity:

- Model complexity adapts to data
- Automatic feature selection
- Sparse representation via support vectors

## Comparison with Other Methods:

### Parametric Linear Classifier

$$f(x) = w^T x + b$$

Fixed  $d + 1$  parameters regardless of training set size.

### Non-parametric SVM

$$f(x) = \sum_{i=1}^{n_{SV}} \alpha_i y_i K(x_i, x) + b$$

$n_{SV}$  support vectors determined by data.

## Benefits of Non-parametric Approach:

- **Flexibility:** No assumptions about decision boundary shape
- **Universality:** Can approximate any function (with appropriate kernel)
- **Robustness:** Less sensitive to model specification

# Kernel Functions and Function Spaces

## Reproducing Kernel Hilbert Space (RKHS):

### Mathematical Framework

Kernel  $K$  defines an infinite-dimensional feature space  $\mathcal{H}$  where linear methods become non-linear in original space.

### Key Properties:

$$\phi : \mathcal{X} \rightarrow \mathcal{H} \quad (56)$$

$$K(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}} \quad (57)$$

### Universal Approximation:

- RBF kernels are universal approximators
- Can represent any continuous function
- Given sufficient training data

### Non-parametric Power

Kernel methods can learn arbitrarily complex decision boundaries without specifying the form in advance.

### Practical Implications:

#### Model Selection Strategy:

- **Start Simple:** Linear kernel first
- **Add Complexity:** Polynomial → RBF
- **Cross-validate:** Choose optimal kernel and parameters

#### Trade-offs:

##### Parametric Advantage

- Fast training and prediction
- Lower memory requirements
- Better interpretability

##### Non-parametric Advantage

- Higher representational power
- Better fit to complex data
- Fewer modeling assumptions

#### When to Use Kernel Methods:

- Non-linear relationships in data

# Summary and Key Takeaways

## Core Concepts Learned:

- **Kernel Trick:** Implicit high-dimensional mapping
- **Support Vectors:** Sparse representation
- **Margin Maximization:** Generalization principle
- **Non-linear Separation:** Via kernels

## Main Algorithms:

- **SVM:** Classification with maximum margin
- **SVR:** Regression with  $\epsilon$ -insensitive loss
- **Kernel Ridge:** Regularized regression
- **Multi-class:** Extensions to multiple classes

## Key Kernels:

- Linear, Polynomial, RBF, Sigmoid
- Mercer's theorem for validity
- Multiple kernel learning

## Practical Guidelines:

### When to Use Kernel Methods

- Non-linear relationships in data
- Need for sparse solutions
- Strong theoretical guarantees required
- Medium-sized datasets

### Parameter Selection:

- **C:** Start with 1.0, tune via CV
- **$\gamma$ :** Start with  $\frac{1}{n\_features}$
- **$\epsilon$ :** Start with 0.1 for SVR

### Limitations:

- Computational complexity:  $\mathcal{O}(n^2)$  to  $\mathcal{O}(n^3)$
- Memory requirements: Store kernel matrix
- Parameter sensitivity
- Not suitable for very large datasets

## Next Steps

Explore deep learning for automatic feature learning  
in machine learning