Introduction to Machine Learning

Lecture 13:

Soft-Margin SVM and Kernel Methods

Prof. Subrahmanya Swamy Peruru
Department of Electrical Engineering
IIT Kanpur

Soft-Margin SVM

Robust to outliers

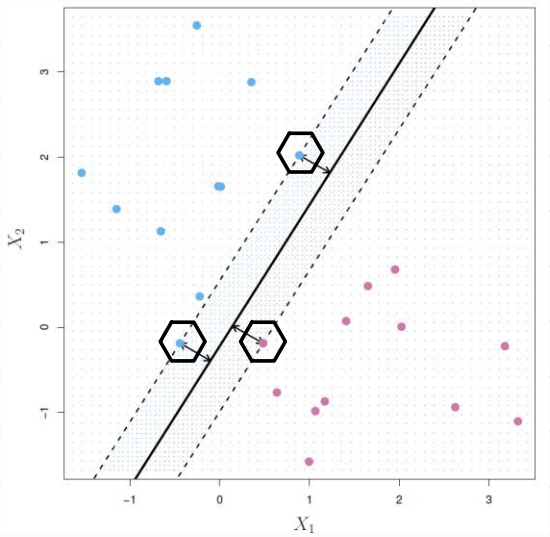
Hard-Margin SVM

Support Vectors: ⬡

Max Margin: $\max \frac{2}{\|\mathbf{w}\|} \quad \Leftrightarrow \quad \min \frac{1}{2} \| \mathbf{w} \|^2$

Hard-margin SVM (primal):

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2} \| \mathbf{w} \|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, n. \end{aligned}$$



Decision rule: $\hat{y}(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$

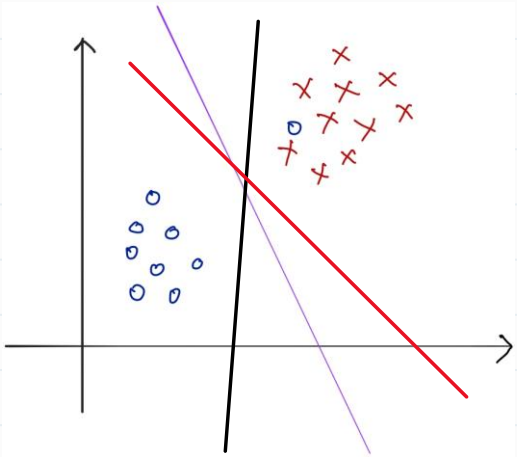
- **Perfect Classification**
- **Strictly no points in margin area**

Support Vectors: $\{(\mathbf{x}_i, y_i) : y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1\}$

Training samples that lie exactly on the margin hyperplanes

Equivalently, the closest points to the decision boundary

When Data Is Not Perfectly Linearly Separable

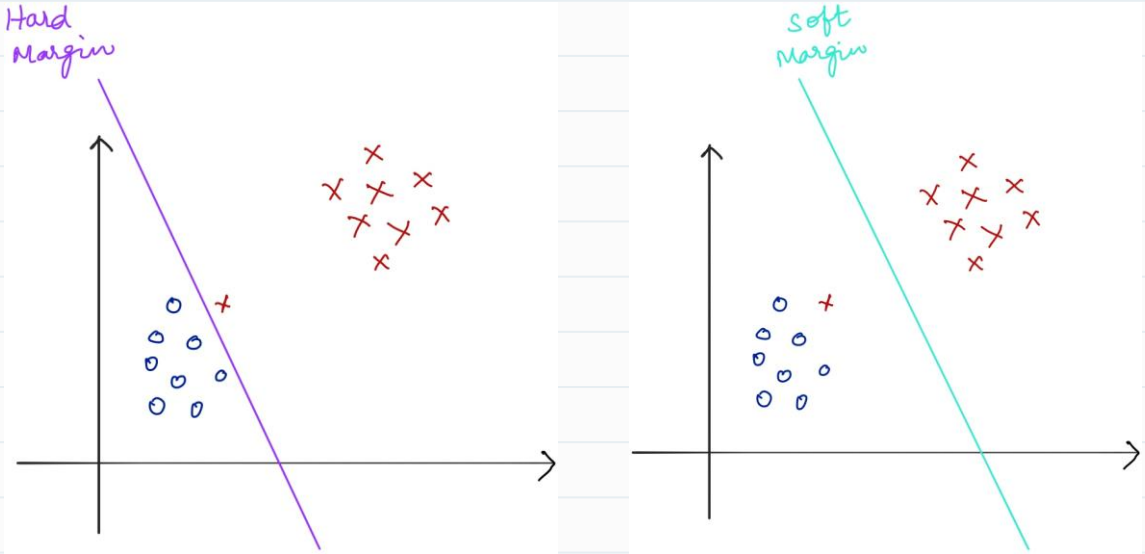


Data is approximately linearly separable

No linear boundary can perfectly separate the data due to outliers

Solution:
Soft-margin SVM, which allows some points to be misclassified due to noise or outliers.

Hard Margin can lead to overfitting

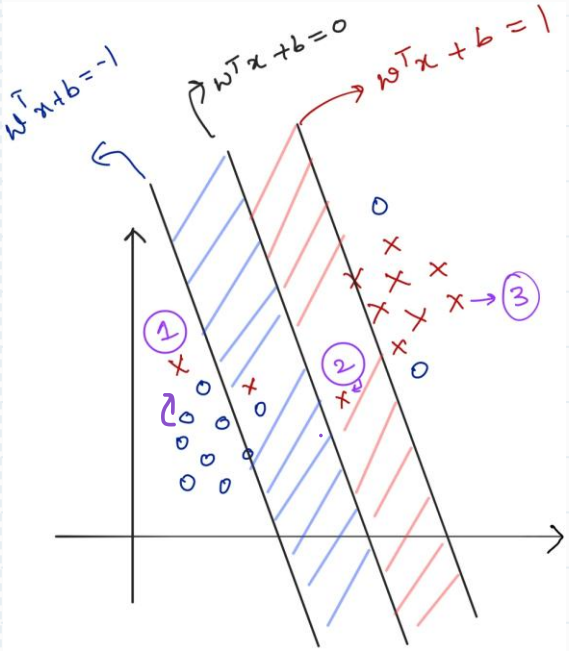


Even with linearly separable training data, hard-margin SVM can overfit due to outliers.

Hard Margin:
- Gives perfect separation
- Sensitive to outliers ⇒ Overfitting

Soft Margin:
- Tolerates misclassification
- More robust to outliers ⇒ Better generalization

Soft-Margin SVM



Allows margin violations (including misclassifications) with **a penalty**

Three types of points:

1. **Misclassification**
   Point lies on wrong side of decision boundary: $y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 0$

2. **Margin violation**
   Correctly classified but lies inside the margin: $0 < y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1$

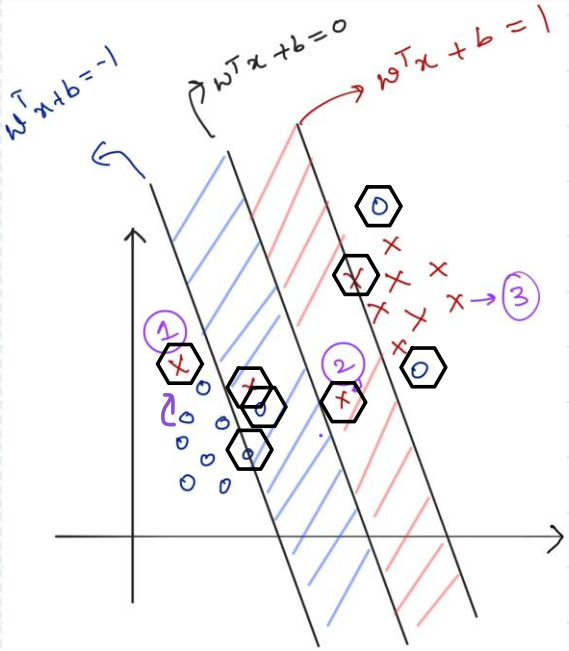
3. **Perfect:**
   Correctly classified and with required margin: $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$

$$\text{penatly} = \begin{cases} 0, & \text{if} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \\ 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b), & \text{if} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1 \end{cases}$$

Compactly, $\boxed{\text{penatly} = \max\{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\}}$

Hinge Loss

Soft-Margin SVM: Hinge Loss Penalty



Hard-margin SVM:

$$\min_{\mathbf{w},b} \quad \frac{1}{2} \| \mathbf{w} \|^2$$
$$\text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, n.$$

In soft-margin, we don't put strict margin constraints;
Instead, we penalize the margin violations using hinge loss.

$$\text{penatly for } i^{\text{th}} \text{ sample} = \max\{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\}$$

Soft-margin SVM:

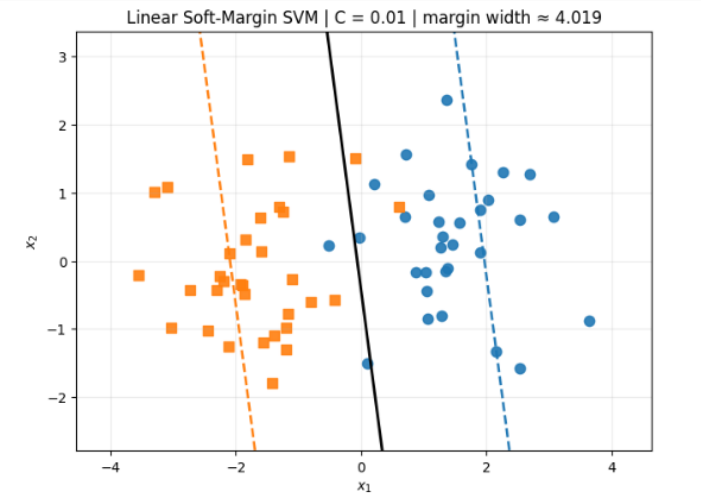
$$\min_{\mathbf{w},b} \quad \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} \max\{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\}$$

Support Vectors:

- Points on the margin hyperplanes
- Points inside the margin area
- Points on the wrong side of the decision boundary

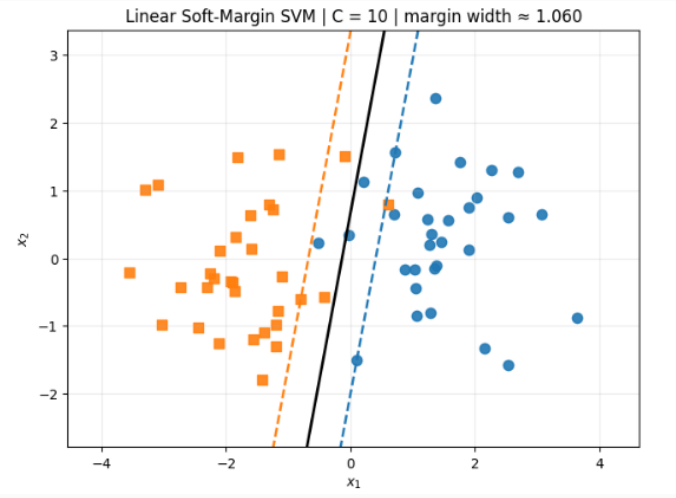
Penalty (Regularization) parameter C

Small C



Large margin, more margin violations

Large C



Small margin, fewer margin violations

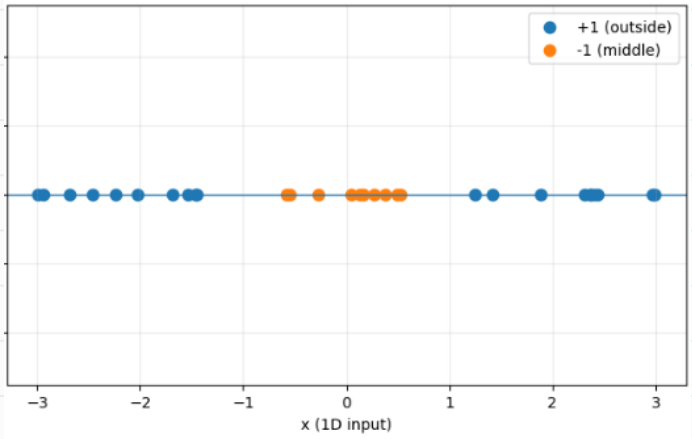
Demo

- C balances margin size and hinge-loss penalty.
- Too small → underfitting; too large → sensitive to noise.
- Choose C via **cross-validation**.

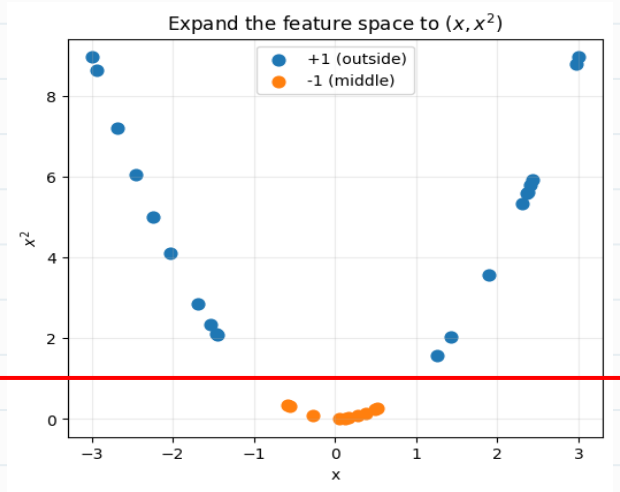
Kernel SVM

Non-linear decision boundaries via kernels

When data is not linearly separable



Can’t separate using a linear hyperplane

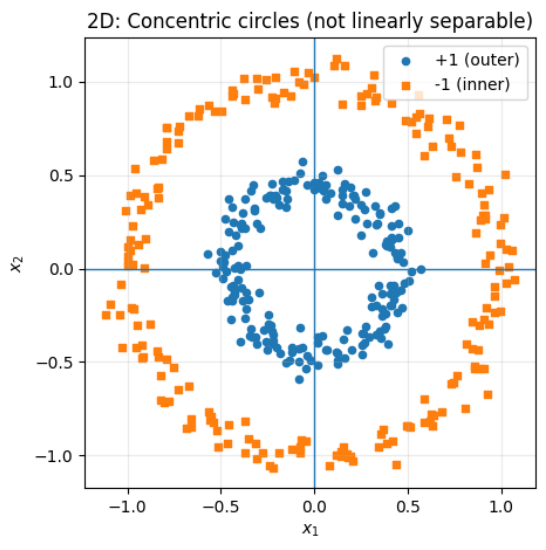


Classes are now linearly separable in the two-dimensional space

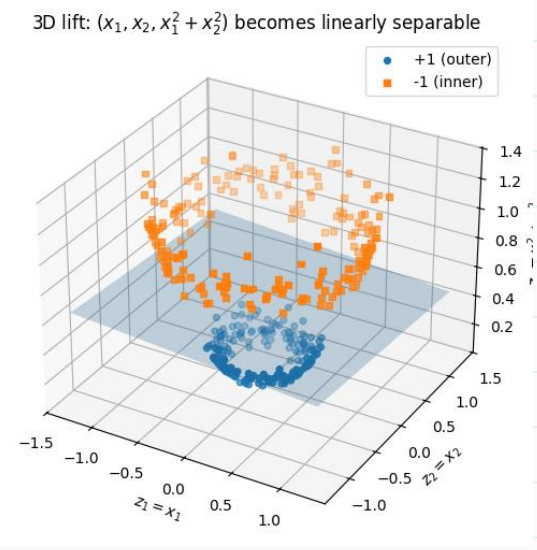
Expand feature space by mapping x to two-dimensions

$$x \to \phi(x) = (\phi_1(x), \phi_2(x)) = (x, x^2)$$

When data is not linearly separable



Can’t separate using a linear hyperplane

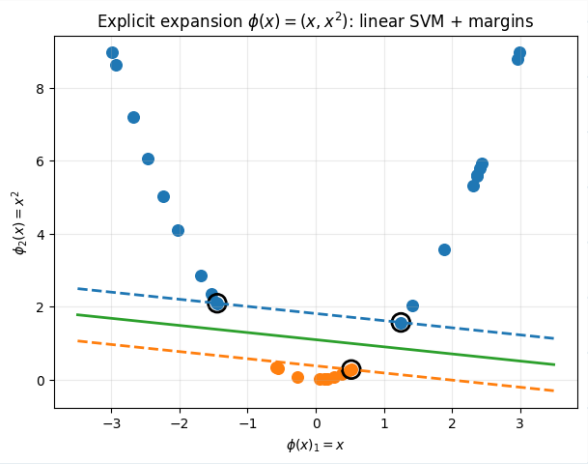


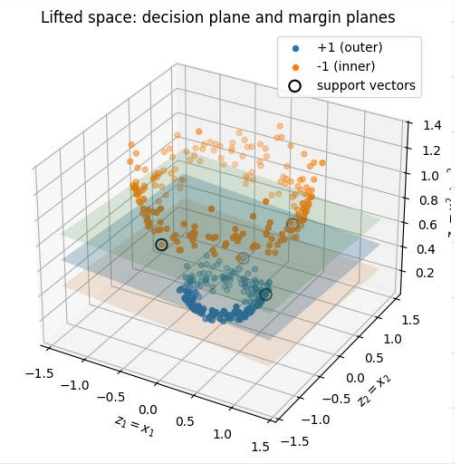
Feature expansion to 3-dimensional space

Input space: $d = 2$
New Feature space: $D = 3$

$$\phi: \mathbb{R}^d \rightarrow \mathbb{R}^D$$
$$(x_1, x_2) \rightarrow (x_1, x_2, x_1^2 + x_2^2)$$

Naive approach for feature expansion





Naive approach:

- Transform the inputs into higher dimensions $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^D,\ \mathbf{x} \rightarrow \phi(\mathbf{x})$
- Explicitly fit SVM in new high-dimensional feature space $\mathbb{R}^D$

$$\min_{\mathbf{w},b} \quad \frac{1}{2} \| \mathbf{w} \|^2$$
$$\text{s.t.} \quad y_i(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b) \geq 1, \quad i = 1, \ldots, n.$$

This approach does not scale well

Issues with explicit (naive) feature expansion

Polynomial Basis expansion: Allows higher order interactions among features up to a certain degree

Example: up to Degree-2 polynomial expansion:

1 feature: $x \to (1,\ x,\ x^2) \Rightarrow 3$ new features

3 features: $(x_1, x_2, x_3) \to (1,\ x_1,\ x_2,\ x_3,\ x_1x_2,\ x_2x_3,\ x_1x_3,\ x_1^2,\ x_2^2,\ x_3^2) \Rightarrow 10$ new features

10 features → degree-5 polynomial expansion → 3003 new features

Gaussian (radial basis) features: Correspond to **infinitely many** polynomial-like features

$$1,\ x_i,\ x_ix_j,\ x_i^2,\ x_ix_jx_k,\ \ldots$$

for **all orders**, not just up to a fixed degree.

Take away:

- Often nonlinear feature expansions can result in very high or infinite-dimensional feature spaces
- Explicitly storing and optimizing in such spaces is computationally expensive.

Goal: Do the same learning without explicitly computing new features $\phi(\mathbf{x})$. (Dual SVM)

Equivalent (Dual) Form of SVM

The SVM optimization problem (**primal form**) admits an equivalent and useful formulation (**dual problem**):

$$\min_{\mathbf{w},b} \quad \frac{1}{2} \| \mathbf{w} \|^2$$
$$\text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \qquad i = 1, \ldots, n.$$

$$\equiv$$

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \, (\mathbf{x}_i^\top \mathbf{x}_j)$$
$$\text{s.t.} \quad \alpha_i \geq 0, \;\; i = 1, \ldots, n, \qquad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

The dual solution $\boldsymbol{\alpha}^\star$, determines the primal solution and the SVM classifier as:

$$\mathbf{w}^\star = \sum_{i=1}^{n} \alpha_i^\star \, y_i \mathbf{x}_i \quad \Rightarrow \quad f(\mathbf{x}) = \mathbf{w}^{\star\top} \mathbf{x} + b = \sum_{i=1}^{n} \alpha_i^\star \, y_i \, (\mathbf{x}_i^\top \mathbf{x}) + b.$$

Key observation: **(The core idea of Kernel SVM)**

The dual problem depends on the input vectors only through inner products $\mathbf{x}_i^\top \mathbf{x}_j$

The final SVM classifier also depends only through $\mathbf{x}_i^\top \mathbf{x}$.

SVM Dual in original and new features

The SVM dual in the original low-dimensional input space and the high-dimensional expanded feature space:

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i \; - \; \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \, \alpha_j \, y_i y_j \, (\mathbf{x}_i^\top \mathbf{x}_j)$$

$$\text{s.t.} \quad \alpha_i \geq 0, \;\; i = 1, \ldots, n, \qquad \sum_{i=1}^{n} \alpha_i \, y_i = 0.$$

Feature expansion →

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i \; - \; \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \, \alpha_j \, y_i y_j \, \left(\boldsymbol{\phi}(\mathbf{x}_i)^{\mathrm{T}} \, \boldsymbol{\phi}(\mathbf{x}_j)\right)$$

$$\text{s.t.} \quad \alpha_i \geq 0, \;\; i = 1, \ldots, n, \qquad \sum_{i=1}^{n} \alpha_i \, y_i = 0.$$

The only difference is $\mathbf{x}_i^\top \mathbf{x}_j$ vs $\boldsymbol{\phi}(\mathbf{x}_i)^T \, \boldsymbol{\phi}(\mathbf{x}_j)$

Example: 10 features → degree-5 polynomial expansion → 3003 new features

In general, $\mathbf{x} \in \mathbb{R}^d$ while $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^D$, where D can be **very large** $(D \gg d)$ **or even infinite**.

Computing an inner product between vectors of length ℓ costs $\mathcal{O}(\ell)$. Hence, $\mathbf{x}_i^\top \mathbf{x}_j = \mathcal{O}(d)$,while $\boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j) = \mathcal{O}(D)$.

Kernel Trick: Efficiently computes $\boldsymbol{\phi}(\mathbf{x}_i)^T \, \boldsymbol{\phi}(\mathbf{x}_j)$ without actually constructing $\boldsymbol{\phi}(\mathbf{x}_j)$

The Kernel Trick

For certain feature expansions, inner products in the expanded feature space $\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{z})$ can be computed directly—without explicitly computing the expanded feature vectors $\boldsymbol{\phi}(\mathbf{x}),\ \boldsymbol{\phi}(\mathbf{z})$

2-Dimensional Input vectors: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \in \mathbb{R}^2$

Polynomial kernel (degree $= 2$):

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + 1)^2 = \phi(\mathbf{x})^\top \phi(\mathbf{z})$$

6-Dimensional feature expansion (explicitly written):

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2}\, x_1 \\ \sqrt{2}\, x_2 \\ x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{bmatrix} \in \mathbb{R}^6, \qquad \boldsymbol{\phi}(\mathbf{z}) = \begin{bmatrix} 1 \\ \sqrt{2}\, z_1 \\ \sqrt{2}\, z_2 \\ z_1^2 \\ \sqrt{2}\, z_1 z_2 \\ z_2^2 \end{bmatrix}$$

Inner product in feature space:

$$\begin{aligned} \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{z}) &= 1 + 2\, x_1 z_1 + 2\, x_2 z_2 + x_1^2\, z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2\, z_2^2 \\ &= 1 + 2(x_1 z_1 + x_2 z_2) + (x_1 z_1 + x_2 z_2)^2 \\ &= (\mathbf{x}^\top \mathbf{z} + 1)^2 \end{aligned}$$

Inner product in the expanded space is computed solely from the inner product in the original space!

Popular kernels and their implied feature spaces

Linear kernel (original feature space)

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z} \quad \Leftrightarrow \quad \phi(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^d$$

Polynomial kernel (degree p)

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + c)^p$$

$$\Leftrightarrow \quad \phi(\mathbf{x}) \text{ contains all monomials of degree } \leq p$$

$$\dim(\phi) = \binom{d+p}{p}$$

Homogeneous polynomial kernel

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^p$$ (only monomials of *exact* degree p appear)

RBF (Gaussian) kernel (corresponds to an infinite-dimensional space)

$$k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma \parallel \mathbf{x} - \mathbf{z} \parallel^2)$$

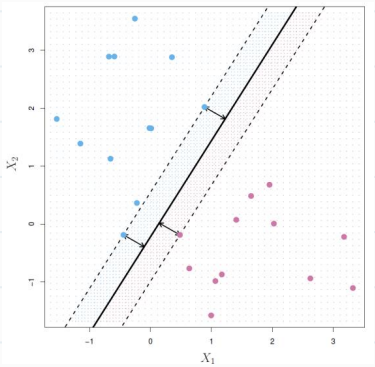
Sigmoid kernel (neural-network-like feature map – a valid kernel only for certain choices of parameters)

$$k(\mathbf{x}, \mathbf{z}) = \tanh(\alpha\, \mathbf{x}^\top \mathbf{z} + c)$$

Key message:

- SVM optimization depends only on inner products;
- kernels implicitly compute inner products in a higher-dimensional feature space.

Support Vector Interpretation



The weight vector is a linear combination of the input vectors:

$$\mathbf{w}^\star = \sum_{i=1}^{n} \alpha_i^\star \, y_i \mathbf{x}_i$$

Only certain coefficients of this linear combination are non-zero.

Those vectors are called support vectors (important points) since weights are entirely determined by those inputs.

Hard-margin support vectors: $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$

Soft-margin support vectors: $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \leq 1$

This fact is proven using KKT Conditions (Complementary slackness)

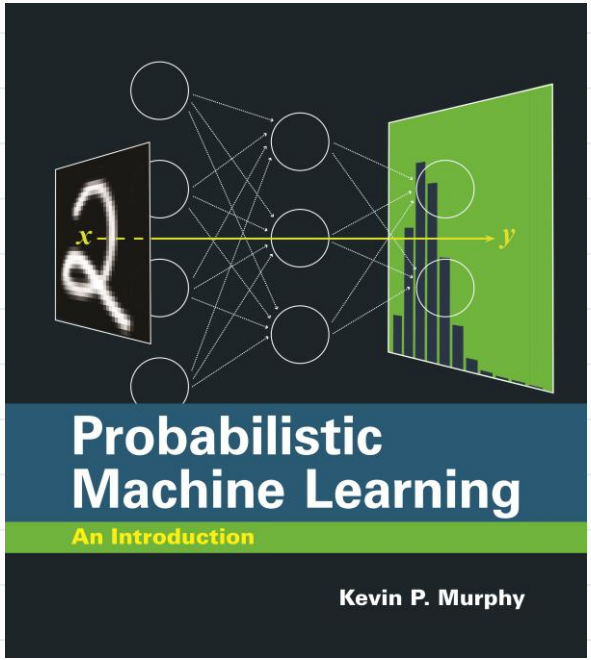
SVM is robust because the weights depend only on support vectors!

Further Reading

- Multi-Class Classifications using multiple Binary classifiers
  - One Vs Rest, One Vs One
- Convex Duality and KKT Conditions
- Application of the Kernel techniques for Linear Regression
- Dual form of Soft-Max SVM

Reference

Chapter 17 of “Probabilistic Machine Learning: An Introduction” book by Kevin P. Murphy



Thank You