

# Support Vector Machines (SVM) and Kernels 1

---

## 1. Classification Methods

### Objective

To understand how different classification algorithms perform — especially on **non-separable but linear data**, i.e., when the classes overlap and a perfect linear boundary does not exist.

---



### Background

In classification, we aim to find a **decision boundary** (or rule) that separates data points from different classes.

However, not all datasets are:

- **Perfectly linearly separable** (can be separated by a straight line or hyperplane)
- Or even **linearly related** (some require nonlinear separation).

Hence, different algorithms handle **overlap** and **nonlinearity** differently.

---



### Question

Arrange the following in the order of decreasing suitability for classifying a non-separable but linear dataset:

1. **Pocket Algorithm (PLA variant with misclassifications allowed)**
2. **LDA (Linear Discriminant Analysis)**
3. **PLA (Perceptron Learning Algorithm)**

## 4. Soft Margin Support Vector Classifier (SVC)

## 5. KNN (k-Nearest Neighbors)

---

### Conceptual Comparison

Algorithm	Nature	Handles Non-Separable Data?	Linear/Non-Linear?	Notes / Suitability
<b>Soft Margin SVC</b>	Margin-based classifier (extension of hard-margin SVM)	<input checked="" type="checkbox"/> Yes (allows some misclassifications)	Linear boundary (extendable to nonlinear with kernels)	<b>Best suited</b> for non-separable but linear data. Trades off margin size vs. errors using penalty parameter ( C ).
<b>Pocket Algorithm</b>	Variant of PLA	<input checked="" type="checkbox"/> Yes (stores best solution so far)	Linear	Works better than PLA on noisy data but less robust than SVC.
<b>LDA (Linear Discriminant Analysis)</b>	Probabilistic (assumes Gaussian class distribution)	<input checked="" type="checkbox"/> Partially (works if overlap isn't too large)	Linear	Performs decently for moderately non-separable data, but sensitive to assumptions.
<b>PLA (Perceptron Learning Algorithm)</b>	Linear classifier (no probability model)	<input type="checkbox"/> No (fails if data not linearly separable)	Linear	Will not converge if data overlap exists.
<b>KNN (k-Nearest Neighbors)</b>	Instance-based (non-parametric)	<input checked="" type="checkbox"/> Yes (no need for separability)	Nonlinear decision boundary	Works for non-linear cases, but doesn't model a linear boundary; less suited when

Algorithm	Nature	Handles Non-Separable Data?	Linear/Non-Linear?	Notes / Suitability
				"linear" is specified.

## Order of Suitability (for non-separable but linear data)

- 1 Soft Margin SVC
- 2 Pocket Algorithm
- 3 LDA
- 4 PLA
- 5 KNN

## Why this Order?

- **Soft Margin SVC:** Allows misclassified points (using slack variables) → best generalization.
- **Pocket Algorithm:** Improves upon PLA by storing the best weights despite misclassifications.
- **LDA:** Works well under the Gaussian class assumption but is not robust to strong overlap.
- **PLA:** Only converges for linearly separable data → fails otherwise.
- **KNN:** Not linear; decision boundary is irregular → not suitable when linearity is required.

## Key Takeaway

When data are **not perfectly separable**, we prefer models that:

- **Allow some violations (soft constraints)**
- **Generalize better**

- **Maintain linear interpretability**

Thus, **Soft Margin SVC** is the most effective linear classifier in this case.

---

## 2. Summary of SVM Concepts

This section introduces the core ideas behind Support Vector Machines (SVMs), why they work, and how they handle both linear and non-linear classification.

---

### ◆ 2.1 Maximal Margin Classifier (Hard-Margin SVM)

#### Goal

Find a separating hyperplane that:

- Correctly separates the classes
- Maximizes the **margin** (distance between the boundary and the closest data points)

#### Key Concepts

- **Hyperplane:**

$$( w^T x + b = 0 )$$

- **Margin:**

Distance between the hyperplane and the nearest data points (**support vectors**).

- **Objective:**

$$\text{Maximize margin} \Leftrightarrow \text{Minimize} ( ||w||^2 )$$



#### Intuition

A larger margin = better generalization.

**BUT**, this only works when data are perfectly separable.

---

### ◆ 2.2 Need for Soft-Margin SVM

Real-world datasets are:

- Noisy
- Overlapping
- Rarely perfectly separable

So hard-margin SVM fails.

**Soft-margin introduces:**

✓ **Slack variables ( $\xi_i$ )**

Allow points to:

- Lie inside the margin, or
- Even be misclassified

✓ **Penalty term ( C )**

Balances:

- Large margin (small ( $\|w\|^2$ ))
- Fewer violations (small slack variables)

---

## ◆ **2.3 Soft-Margin Scenarios**

Soft margin allows two types of "violations":

1. **Correct side, but inside the margin**
2. **Wrong side of the hyperplane (misclassified)**

This keeps the classifier:

- Robust
- Able to handle non-separable data
- Resistant to outliers and noise

---

## ◆ **2.4 Why Soft-Margin Helps**

## Advantages

- **Better convergence** than PLA/Hard SVM
  - **Less overfitting** because the margin is maximized with flexibility
  - **Better noise handling**
  - Still keeps a **linear decision boundary**
- 

## 2.5 Motivation for Non-Linear SVM

Sometimes the true boundary is **not linear**.

Examples from the notes:

- Tyre pressure (too low or too high is bad → U-shaped)
- Medicine dosage (low & high ineffective → nonlinear)

A linear separator cannot solve such problems.

**Idea:**

**Transform the data to a higher-dimensional feature space**

where a linear separator *does* exist.

This leads to the next stage: **kernels**.

---

## 2.6 Non-linear Transformation

**Approach:**

Use a transformation function

$$\phi(x) \rightarrow \text{higher-dimensional feature space}$$

(e.g., adding  $(x^2)$ ,  $(x_1x_2)$ , etc.)

After transformation, data may become linearly separable.

But...

 **Explicit feature mapping becomes:**

- Very high-dimensional
  - Computationally expensive
  - Hard to design manually
- 

## ◆ 2.7 Enter Kernels (Key Insight)

Instead of computing:

$$\phi(x_i) \quad \text{and} \quad \phi(x_j)$$

use a **kernel function**:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

This computes the **inner product in the high-dimensional space**  
**without ever explicitly computing the transformation.**

🎉 This is the **Kernel Trick**

---

## ◆ 2.8 Summary of this Summary (quick notes)

- Hard-margin → for perfect separability
  - Soft-margin → allows violations using slack variables
  - (C) controls margin vs misclassification tradeoff
  - Nonlinear boundaries require feature transformation
  - Kernels enable high-dimensional boundaries efficiently
  - SVMs are powerful due to margins + kernels + convex optimization
- 

## 3. Hard-Margin Support Vector Classifier (SVC)

The hard-margin SVC is the **original** version of SVM designed for **perfectly linearly separable data**.

It finds the *widest possible margin* while ensuring **zero misclassifications**.

---

## ◆ 3.1 Problem Setting

We are given a dataset:

- Features: ( $x_i \in \mathbb{R}^d$ )
- Labels: ( $y_i \in +1, -1$ )

We want a hyperplane:

$$w^T x + b = 0$$

that **perfectly separates** the positive and negative classes.

---

## ◆ 3.2 Margin Definition

✓ **Distance of a point ( $x_i$ ) from the hyperplane:**

$$\text{dist}(x_i) = \frac{y_i(w^T x_i + b)}{|w|}$$

✓ **Margin**

Distance between the hyperplane and the closest data points.

We want this distance to be **as large as possible**.

---

## ◆ 3.3 Hard-Margin Constraints

For perfect separation:

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

This ensures:

- Points are **correctly classified**
  - Points lie **outside** the margin
-

## ◆ 3.4 Optimization Objective

### ✓ Maximize the margin

$$\text{Margin} = \frac{2}{|w|}$$

Maximizing margin  $\Leftrightarrow$  Minimizing  $|w|$

Thus, the optimization problem becomes:

$$\min_{w,b} \frac{1}{2} |w|^2$$

subject to:

$$y_i(w^T x_i + b) \geq 1$$

## ◆ 3.5 Geometric Interpretation

### ◆ What is the model doing?

- It draws a hyperplane **between** classes.
- It finds two parallel boundary lines:

$$w^T x + b = +1 \quad w^T x + b = -1$$

- The distance between these lines = **margin**.
- Only a few points touch these margins  $\rightarrow$  **support vectors**.

### ◆ Why minimize ( $|w|^2$ )?

The vector ( $w$ ) controls:

- Orientation of the hyperplane
- **Thickness of the margin slab**

Smaller ( $|w|$ )  $\rightarrow$  larger margin.

## ◆ 3.6 When Hard-Margin Works Well

- ✓ Data perfectly separable
- ✓ No noise
- ✓ No overlapping classes
- ✓ Very low outliers

## ✖ Problems

- Real-world data rarely satisfies these conditions
- Sensitive to outliers (one mislabeled point breaks everything)
- Fails when data is non-separable

This is why **Soft-Margin SVM** was introduced.

---

## ◆ 3.7 Key Takeaways

- Hard margin = strict, no errors allowed
  - Optimizes for *maximum* margin
  - Overfits easily if even one point violates separability
  - Great for synthetic or clean academic datasets
  - Not robust → replaced by Soft-Margin in most practical scenarios
- 

# 4. Soft-Margin Support Vector Classifier (SVC)

Soft-Margin SVM is the **practical**, real-world version of SVM.

It fixes the limitations of the hard-margin SVM by allowing:

- some misclassification
  - some points inside the margin
  - better tolerance to noise & overlap
-

## ◆ 4.1 Why Do We Need Soft-Margin SVM?

Hard-margin SVM requires:

- no noise
- perfect linear separability
- no point allowed inside the margin

But real datasets often have:

- mislabeled points
- overlapping distributions
- outliers
- non-separable clusters

So the hard constraint must be relaxed.

Soft-Margin SVM introduces **controlled violations**.

---

## ★ 4.2 Slack Variables ( $\xi_i$ )

Slack variables ( $\xi_i \geq 0$ ) measure how much a point violates the margin conditions.

There are 3 possibilities:

---

### ✓ 1. Point outside the margin (correctly classified)

$$\xi_i = 0 \quad y_i(w^T x_i + b) \geq 1$$

---

### ✓ 2. Point inside the margin but still correctly classified

$$0 < \xi_i < 1 \quad 0 < y_i(w^T x_i + b) < 1$$

This means:

- On the correct side
  - But closer to the hyperplane than ideal
-

### ✓ 3. Misclassified point (wrong side of the hyperplane)

$$\xi_i > 1 \quad y_i(w^T x_i + b) < 0$$

These are *true violations*.

---

## ◆ 4.3 Soft-Margin Optimization Problem

We want to:

- keep the margin large
- keep violations small

So the objective becomes:

$$\min_{w,b,\xi} \frac{1}{2} |w|^2 + C \sum_{i=1}^N \xi_i$$

Subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

---

## ◆ 4.4 Role of the Penalty Parameter (C)

The hyperparameter **C** balances:

margin width vs. number of violations

---

### ✓ High C (strict, less tolerance)

- Slack must be small
- Few violations allowed
- Classifier becomes **rigid**
- Risk of **overfitting**

Used when:

- low noise

- want fewer misclassifications
- 

### ✓ **Low C (lenient, more tolerance)**

- Violations allowed
- Margin can expand
- More flexible
- Reduces overfitting
- Better for noisy data

Used when:

- overlapping classes
  - noisy labels
  - robust generalization is desired
- 

## ◆ **4.5 Visual Intuition**

Soft-margin SVM allows:

- points in the margin
- some on the wrong side
- as long as the collective penalty is reasonable

This creates a **thicker, stable margin** that generalizes better.

---

## ◆ **4.6 Advantages of Soft-Margin SVM**

- ✓ Works for **non-separable** datasets
- ✓ Robust to noise
- ✓ Allows misclassifications
- ✓ Better generalization than hard-margin
- ✓ Still keeps a **linear** decision boundary

✓ Foundation for **kernel SVMs**

---

## ◆ **4.7 When to Use Soft-Margin SVM**

Soft-Margin SVM is ideal when:

- data is mostly linear but not perfect
- small amount of overlap exists
- you want stability + generalization
- some noise/mislabeled points exist

In practice:

👉 Almost all linear SVMs used in industry are soft-margin SVMs.

---

## ◆ **4.8 Key Idea Summary (quick notes)**

- Introduces **slack variables** ( $\xi_i$ )
- Allows **controlled misclassifications**
- Optimization:

$$\min \frac{1}{2} |w|^2 + C \sum \xi_i$$

- Hard margin = infinite C
- Soft margin = finite C
- High C  $\rightarrow$  strict, overfit
- Low C  $\rightarrow$  flexible, better generalization

## **5. Non-Linearly Separable Data**

This section explains *why* linear SVMs fail on certain real-world problems and motivates the use of kernels.

---

## ◆ **5.1 What Does “Non-Linearly Separable” Mean?**

A dataset is **linearly separable** if there exists a straight line (in 2D) or hyperplane (in higher dimensions) that separates the classes perfectly.

But many datasets do NOT satisfy this.

Examples:

- Circular patterns
- Concentric rings
- XOR pattern
- U-shaped decision boundaries

In such cases, **no straight line can separate the classes**.

---

## ◆ **5.2 Real-World Motivating Examples (from professor's notes)**

Your professor gives intuitive real-world cases where the decision boundary is not linear.

### 🟡 **Example 1: Tyre Pressure**

- Very low pressure → bad performance
- Very high pressure → also bad
- Middle range → good

This forms a **U-shaped** or **inverted U-shaped** pattern.

A straight line can't separate "good" from "bad".

---

### 🟡 **Example 2: Medicine Dosage**

- Very low dose → ineffective
- Very high dose → risky/harmful
- Moderate dose → ideal

Again, a **non-linear pattern**:

- Bad → Good → Bad

A linear boundary cannot model this.

---

## ◆ 5.3 Why Linear SVM Fails Here

Standard SVM (without kernels) creates a **linear boundary**:

$$w^T x + b = 0$$

If the dataset has **curved** or **complex** boundaries:

- Linear SVM forces a straight line
- The result is **poor classification**
- Margin may be meaningless
- Soft margin helps only slightly—cannot fix fundamental nonlinearity

Therefore, SVM must operate in a **higher-dimensional space** where the classes *become* linearly separable.

---

## ◆ 5.4 The Key Insight

★ If the data is not linearly separable in the original space,

It might be separable in a higher-dimensional feature space.

Example:

- Data in  $\mathbb{R}^2$  cannot be separated
- But mapping to  $\mathbb{R}^3$ :

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

creates a plane that separates the classes.

This is the principle behind:

- Polynomial kernels
  - Gaussian RBF kernels
  - Other non-linear kernels
-

## ◆ 5.5 The Challenge With Explicit Feature Mapping

Directly computing these transformations is often:

- ✗ computationally expensive
- ✗ high-dimensional (even infinite-dimensional)
- ✗ hard to design manually
- ✗ slow for large datasets

We need a smarter way to use high-dimensional features without explicitly calculating them.

This leads to the **kernel trick**, which is covered next.

---

## ◆ 5.6 Key Takeaways

- Many real-world problems have **non-linear** patterns.
  - Linear SVM, even with soft margin, cannot solve these effectively.
  - Mapping data to a **higher dimension** can make it linearly separable.
  - But direct mapping is expensive → motivates **kernels**.
- 

# 6. Kernel Trick and Kernel Methods

This section explains **how SVMs handle non-linear data** without explicitly transforming features into high dimensions.

This is what makes SVMs so powerful.

---

## ◆ 6.1 Motivation: Non-Linear Boundaries Need More Features

When data has a curved or complex boundary, we need to map it to a **higher-dimensional space** where a linear separator exists.

Example:

XOR pattern → not separable in 2D

Add a new dimension → separable in 3D

But doing this manually is:

- slow
- computationally expensive
- may require thousands or infinite features
- hard to design

We need a smarter way.

---

## ★ 6.2 Key Insight: SVM Uses Only Inner Products

In the **dual SVM formulation**, all calculations involve only inner products:

$$x_i \cdot x_j$$

The classifier is expressed as:

$$f(x) = \sum_i \alpha_i y_i (x_i \cdot x)$$

So, instead of working directly with  $\phi(x)$  (feature maps), we only need:

$$\phi(x_i) \cdot \phi(x_j)$$

This is the foundation of kernels.

---

## ★ 6.3 Kernel Function (Definition)

A **kernel** is a function:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

It computes the **inner product in the high-dimensional feature space without ever computing  $\phi(x)$  explicitly**.

This is the **Kernel Trick**.

---

## ◆ 6.4 Why the Kernel Trick Is So Powerful

### ✓ No need to compute high-dimensional transformations

Instead of calculating  $\phi(x)$  with 1000+ dimensions, we use a simple kernel function in the original space.

### ✓ Saves time & memory

Transforms that were impossible become easy.

### ✓ Enables infinite-dimensional feature spaces

Example: A Gaussian RBF kernel maps to **infinite** dimensions, yet it computes in **constant time**.

### ✓ Non-linearity handled elegantly

Just choose a kernel → instantly non-linear SVM.

---

## ◆ 6.5 Famous Kernel Functions

Your professor mentioned two main ones, but let's list them properly:

### 1. Polynomial Kernel

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d$$

- Degree ( d ) controls complexity
  - Captures curved boundaries
  - Equivalent to using all polynomial combinations of features
- 

### 2. Gaussian RBF Kernel

$$K(x_i, x_j) = \exp(-\gamma|x_i - x_j|^2)$$

- Most widely used kernel
- Maps to infinite-dimensional space

- Parameter  $\gamma$  controls “spread”
    - Large  $\gamma \rightarrow$  tight peaks  $\rightarrow$  overfitting
    - Small  $\gamma \rightarrow$  smooth boundary  $\rightarrow$  underfitting
- 

### 3. Linear Kernel

$$K(x_i, x_j) = x_i \cdot x_j$$

- Same as regular linear SVM
  - Useful for large, sparse datasets (e.g., text classification)
- 

## ◆ 6.6 What Kernel Trick Achieves

### ● Transforms data to a higher dimension implicitly

→ allows linear separation in feature space

### ● Keeps computation in original dimension

→ efficient and scalable

### ● Supports many types of non-linearity

→ circles, spirals, clusters, rings, XOR, etc.

---

## ◆ 6.7 Conceptual Flow of Kernel SVM

1. You give SVM a kernel (e.g., RBF)
2. SVM:
  - Computes pairwise kernel values
  - Builds an optimization problem
  - Finds support vectors
3. New prediction uses:

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

Where:

- Only support vectors matter
  - $\alpha_i$  are learned weights
  - K defines the shape of the boundary
- 

## ◆ 6.8 Why This Works in Practice

- You never compute  $\varphi(x)$
- No large feature matrices
- No direct high-dimensional math
- Just compute kernel values

This is why SVM is:

- Strong
  - Elegant
  - Mathematically sound
  - Often more powerful than neural networks for small-to-medium datasets
- 

## ◆ 6.9 Summary (Quick Notes)

- Kernel = inner product in higher dimension
  - Kernel Trick = avoiding explicit feature expansion
  - SVM depends only on dot products → kernels replace them
  - Popular kernels: Polynomial, RBF, Linear
  - Enables SVM to solve **non-linear** problems
  - RBF kernel is most commonly used
-

# Vapnik's Contribution (in short)

Vladimir Vapnik is the **co-creator of Support Vector Machines (SVMs)**.

His key contributions are:

## 1. The concept of maximizing the margin

Vapnik introduced the idea that the best classifier is the one with the **largest margin**, because a larger margin leads to better generalization.

This is the foundation of SVM theory.

---

## 2. The introduction of the soft-margin formulation

Real-world data aren't perfectly separable.

Vapnik designed the **soft-margin SVM**, which uses:

- slack variables
- penalty parameter ( C )

This made SVM usable in real applications.

---

## 3. The kernel trick (dual formulation insight)

Vapnik showed that in the **dual form** of the SVM optimization problem:

- all computations involve **inner products**
- these inner products can be replaced by **kernels**

This enabled SVMs to work in:

- high-dimensional
  - even infinite-dimensional
- feature spaces **without** explicit transformations.
- 

## 4. Structural Risk Minimization (SRM)

Vapnik developed the concept of **SRM** — a theoretical framework that balances:

- model complexity

- training error

SVM is a direct application of SRM.

This is why SVMs generalize well even with small data.

---

### ★ In one line:

Vapnik transformed the idea of linear margin maximization into a powerful, general, high-dimensional classification framework using soft margins and kernels.

---

## 7. Common Kernel Functions (Polynomial & RBF Kernels)

---

### ◆ 7.1 Why Do We Need Different Kernels?

Different datasets have different shapes of decision boundaries.

- Some need **curved** boundaries
- Some need **highly flexible** boundaries
- Some need **smooth** boundaries
- Some need **simple** linear boundaries

Choosing the right kernel determines:

- How powerful the classifier is
  - How flexible the boundary can be
  - How well does SVM generalize
- 

### ★ 7.2 Polynomial Kernel

#### 📌 Formula

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d$$

Where:

- (d) = degree of the polynomial
  - (c) = constant (usually 0 or 1)
- 

## ◆ Intuition

The polynomial kernel allows the classifier to use *interactions* and *higher-degree* terms of features, for example:

- $(x_1^2), (x_2^2)$
- $(x_1 x_2)$
- $(x_1^3, x_2^3)$

It can model boundaries like:

- curves
  - parabolas
  - cubic shapes
  - more complex polynomial patterns
- 

## ◆ Visualization

Your professor's slide references a YouTube link showing how an increasing **degree** bends the boundary more:

- Degree 1 → straight line
- Degree 2 → parabolic curve
- Degree 3 → more flexible curves

Higher degree → more complexity → more risk of overfitting.

---

## ◆ When to Use Polynomial Kernel

Useful when:

- Interactions between features matter

- Data has a moderately complex boundary
- You want a smooth but non-linear decision region

Common in:

- Image recognition
  - Some bioinformatics datasets
- 

## ⭐ 7.3 Radial Basis Function (RBF) Kernel

### 📌 Formula

$$K(x_i, x_j) = \exp(-\gamma|x_i - x_j|^2)$$

Where:

- ( $\gamma$ ) controls how far a point's influence extends.
- 

### ◆ Intuition

The RBF kernel measures similarity based on distance:

- If two points are **close** → high kernel value
- If **far apart** → kernel value approaches 0

This creates **local bumps** in decision space and allows extremely flexible boundaries.

This kernel is **the most powerful** and most widely used in SVM.

---

### ◆ Effect of Gamma ( $\gamma$ )

#### ✓ Low $\gamma$ (small gamma)

- Large spread
- Smooth boundary
- Underfits if too low
- Very generalized

## ✓ High γ (large gamma)

- Very small spread
- Each point creates a sharp influence
- Highly flexible, tight boundary
- Risk of **overfitting**

Your professor's slide shows:

**Underfit → → Overfit**

as gamma increases.

---

## ◆ Why RBF Kernel Is So Popular

- Works extremely well for most datasets
  - Can model very complex shapes
  - Maps to **infinite-dimensional space**
  - Needs only one hyperparameter ( $\gamma$ )
  - Very stable in practice
- 

## ★ 7.4 Linear Kernel (for completeness)

$$K(x_i, x_j) = x_i \cdot x_j$$

Used when:

- Data is already (almost) linearly separable
  - Number of features is large (text data, NLP)
- 

## ★ 7.5 Choosing Between Polynomial and RBF Kernels

Kernel	Best When	Notes
<b>Polynomial</b>	Data has structured interactions or polynomial trends	Degree grows complexity quickly

Kernel	Best When	Notes
RBF	Most datasets; unknown structure	Safest default choice
Linear	Very high-dimensional data (e.g., text)	Fastest and simplest

## ⭐ 7.6 Key Takeaways

- Polynomial kernel = curved, polynomial boundaries
- RBF kernel = flexible, smooth, infinite-dimensional
- $\gamma$  controls spread (high  $\gamma$  overfits)
- RBF is usually the best default kernel
- Choice of kernel determines the decision boundary shape

## ⭐ Gaussian RBF Kernel

The **Gaussian Radial Basis Function (RBF) kernel** is the **most widely used kernel** in SVM because it can model highly complex and non-linear decision boundaries.

### ✓ Formula

$$K(x_i, x_j) = \exp(-\gamma|x_i - x_j|^2)$$

- ( $K(x_i, x_j)$ ) = similarity between points  $(x_i)$  and  $(x_j)$
- ( $|x_i - x_j|^2$ ) = squared Euclidean distance
- ( $\gamma$ ) = controls how quickly similarity decreases with distance

## ◆ Intuition: What Does the RBF Kernel Do?

The RBF kernel answers one question:

| How similar is point  $(x_j)$  to point  $(x_i)$ ?

- If  $(x_i)$  and  $(x_j)$  are **close**,  
( $|x_i - x_j|^2$ ) is small  $\rightarrow$  kernel  $\approx 1$  (high similarity)
- If they are **far apart**,

( $|x_i - x_j|^2$ ) is large  $\rightarrow$  kernel  $\approx 0$  (low similarity)

Essentially:

- Each support vector creates a "**bump" or "hill of influence**" around itself
  - New points are classified by how strongly they fall under these bumps
- 

## ⭐ How Gamma ( $\gamma$ ) Controls the "Spread" of These Bumps

Gamma determines **how wide or narrow each bump is.**

---

### ➊ 1. Low Gamma ( $\gamma$ small $\rightarrow$ large spread)

**Effect:**

- Each point influences a **large area**
- The similarity function decays **slowly**
- Boundary becomes **smooth** and less sensitive to local variations

**Interpretation:**

- Wide "hills"
- Model is **simple**
- Tends to **underfit** if too low

**Visual Intuition:**

. . . (each point influences a huge region)

~~~~~-----~~~~~

smooth, broad decision regions

### ➋ 2. High Gamma ( $\gamma$ large $\rightarrow$ small spread)

## Effect:

- Each point's influence is **very local**
- Similarity decays **very quickly**
- Boundary becomes **highly flexible**

## Interpretation:

- Very narrow "hills"
- Model becomes sensitive to noise
- Tends to **overfit** if too high

## Visual Intuition:

. . . . (each point creates a sharp peak)

^ ^ ^ ^

sharp, wiggly decision boundary

## ★ Summary: Spread vs. Gamma

| Gamma ( $\gamma$ ) | Spread (Variance) | Decision Boundary | Risk         |
|--------------------|-------------------|-------------------|--------------|
| Low $\gamma$       | Large spread      | Smooth, simple    | Underfitting |
| High $\gamma$      | Small spread      | Complex, sharp    | Overfitting  |

## 🌀 Why This Happens (Math Intuition)

The formula:

$$\exp(-\gamma d^2)$$

behaves differently depending on  $\gamma$ .

- If  $\gamma$  is small  $\rightarrow$  even large distances produce non-zero values  $\rightarrow$  similarity remains high over a large region
- If  $\gamma$  is large  $\rightarrow$  even small distances drop similarity quickly  $\rightarrow$  influence becomes extremely localized

So gamma is basically controlling the **effective radius of influence**.

**More gamma → less radius**

**Less gamma → more radius**

---

## ⭐ Connection to Variance

In probability, the Gaussian distribution:

$$\exp\left(-\frac{d^2}{2\sigma^2}\right)$$

has variance ( $\sigma^2$ ).

Compared to RBF:

$$\gamma = \frac{1}{2\sigma^2}$$

So:

- **Large  $\gamma \rightarrow \text{small } \sigma^2 \rightarrow \text{small spread}$**
- **Small  $\gamma \rightarrow \text{large } \sigma^2 \rightarrow \text{large spread}$**

This is the exact connection your professor refers to as "**spread (variance)**".

---

## 🎉 Final Intuition Summary

- ✓ RBF kernel = similarity measure using distance
  - ✓ Gamma controls how quickly similarity falls off
  - ✓ Low gamma  $\rightarrow$  wide, smooth boundaries
  - ✓ High gamma  $\rightarrow$  tight, wiggly boundaries
  - ✓ Gamma =  $1 / (2 \times \text{variance})$ , so it directly controls spread
-