

Support Vector Machines (SVM) and Kernels 2

★ 8. SVM Dual Formulation

The dual formulation is one of the most important concepts in SVMs.

It transforms the optimization problem into a form that:

- depends only on dot products
- naturally incorporates kernels
- automatically highlights support vectors
- is computationally easier for high-dimensional data

Let's break it down step-by-step.

◆ 8.1 Primal Problem (Original SVM Optimization)

The **primal** optimization problem for a *hard-margin* SVM is:

Objective: Minimize the norm of weights

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

Subject to constraints that enforce correct classification:

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

This means:

- All points must lie **outside** the margin
 - No slack variables (for soft-margin)
 - The boundary is strictly linear
-

◆ 8.2 Why Move to the Dual?

The primary problem has limitations:

1. In high-dimensional spaces, w has too many parameters

- Hard to optimize directly
- SVMs often operate in very high-dimensional or even infinite-dimensional feature spaces

2. We want a way to use kernels

- Kernels depend on inner products between data points
- Primal uses w and b directly \rightarrow cannot insert kernel easily

3. We want a form that naturally identifies support vectors

- In primal, all points influence w
- In dual, only some data points receive non-zero coefficients
 \rightarrow these are the **support vectors**

Because of these advantages, SVMs are almost always solved in the dual.

★ 8.3 Deriving the Dual Problem Using Lagrange Multipliers

To convert constrained optimization \rightarrow unconstrained optimization, we introduce **Lagrange multipliers** ($\alpha_i \geq 0$).

We form the Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 + \sum_i \alpha_i [y_i (w^T x_i + b) - 1]$$

We then:

1. Take partial derivatives with respect to w and b
2. Set them to 0 (stationarity)

3. Substitute back into the Lagrangian
4. Obtain a new optimization problem in terms of **α only**

This new problem is the **dual formulation**.

★ 8.4 The Final Dual Problem

The dual turns out to be:

Maximize

$$\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

Subject to

$$\alpha_i \geq 0$$

$$\sum_i \alpha_i y_i = 0$$

This is now a **quadratic programming problem** in terms of the α 's.

◆ 8.5 Why the Dual Is Powerful

★ 1. Optimization depends only on dot products

All occurrences of the data appear in terms like:

$$x_i \cdot x_j$$

This is crucial because it lets us replace these dot products with **kernels**:

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

This instantly makes the SVM **non-linear** without changing the optimization process.

★ 2. Identifies Support Vectors Automatically

The solution to the dual gives values of (α_i).

- If ($\alpha_i = 0$) \rightarrow point is **not** a support vector
- If ($\alpha_i > 0$) \rightarrow point lies on or inside the margin
 \rightarrow becomes a **support vector**

Thus:

- Only support vectors matter.
- They define the entire decision boundary.

This gives SVMs:

- Compressibility
- Sparsity
- Stability

★ 3. Efficient for High Dimensions

When d (features) is very large:

- In primal: optimizing w of dimension d is expensive
- In dual: we optimize only N variables (α 's), where N = number of points

This is often much easier.

◆ 8.6 Representing the Final Hypothesis Using Support Vectors

Once we solve for the α 's, the SVM classifier becomes:

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

Important points:

- Only **support vectors** (SV) appear because their α 's are non-zero.
- Each support vector contributes a weighted similarity measure.
- The decision boundary is determined by a handful of critical points.

This is one of the most elegant properties of SVMs.

★ 8.7 Incorporating Kernels into the Dual Problem

Wherever SVM uses a dot product, we replace it with a **kernel function**:

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

Thus the dual becomes:

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

And the hypothesis becomes:

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

This allows SVM to operate in a:

- higher
- sometimes infinite-dimensional
- non-linear feature space **without ever computing the space explicitly.**

★ 8.8 Why Kernels Only Work in the Dual and Not Primal

In primal:

- The model uses w explicitly
- w must be represented as a vector in feature space
- We cannot represent w if the feature space is infinite (e.g., RBF kernel)

In dual:

- The model depends only on inner products
- Kernels plug directly into these places
- No explicit feature vectors needed

Thus:

Kernels exist only because of the dual formulation.
Without the dual, kernel SVMs would not be possible.

★ 8.9 Final Summary (Exam-Ready)

- ✓ Primal minimizes ($\frac{1}{2}|w|^2$) with classification constraints
 - ✓ Dual uses Lagrange multipliers to turn constraints into a quadratic optimization problem
 - ✓ Dual expresses everything using inner products ($x_i \cdot x_j$)
 - ✓ Support vectors naturally emerge as $\alpha_i > 0$
 - ✓ Kernels replace dot products → enabling non-linear SVM
 - ✓ Final decision rule uses only support vectors
 - ✓ Dual is computationally more efficient for high dimensions
-

★ 9. Multi-Class Classification in SVM

Standard SVMs are **binary classifiers** → they naturally handle only **two classes** (e.g., +1 vs -1).

But many problems have more than two classes:

- Handwritten digits (0–9)
- Types of flowers (setosa, versicolor, virginica)
- Emotions, animals, disease categories, etc.

Since SVM doesn't *natively* extend to multi-class, we use **strategies** to build multiple binary SVM classifiers and combine them.

The two most common methods are:

1. **One-vs-Rest (OvR)**
 2. **One-vs-One (OvO)**
-

★ 9.1 One-vs-Rest (OvR) Approach

Also called **One-vs-All (OvA)**.

◆ Idea

For **K classes**, train **K SVM models**.

Each model tries to separate:

Class k vs All other classes

For example, in a 4-class problem:

- Model 1: Class 1 vs {2,3,4}
- Model 2: Class 2 vs {1,3,4}
- Model 3: Class 3 vs {1,2,4}
- Model 4: Class 4 vs {1,2,3}

Each model is a **binary classifier**.

◆ Training Process

For each model k :

- Label all samples in class $k \rightarrow +1$
- Label all other samples $\rightarrow -1$
- Train a binary SVM

So we end up with **K decision boundaries**.

◆ Prediction

A test sample (x) is fed into all K SVMs.

Each SVM outputs a **decision score**:

$$f_k(x) = w_k^T x + b_k$$

The sample is assigned to the class with the **highest score**.

Formally:

$$\hat{y} = \arg \max_k f_k(x)$$

◆ Advantages of OvR

- ✓ Simple to implement
- ✓ Requires only **K models** (efficient for large K)

✓ Works well when:

- Classes are well separated
 - Training data per class is balanced
-

◆ Disadvantages of OvR

- ✗ Negative class is much larger than positive class → class imbalance
 - ✗ Some SVMs may produce ambiguous output (two can predict positive)
 - ✗ Decision boundaries may overlap or be inconsistent
 - ✗ Not always the best for tightly clustered multi-class data
-

★ 9.2 One-vs-One (OvO) Approach

Also called **Pairwise Classification**.

◆ Idea

For **K classes**, train an SVM for **every pair of classes**.

Number of classifiers required:

$$\frac{K(K-1)}{2}$$

Example: For 4 classes → 6 classifiers

- (1 vs 2)
- (1 vs 3)
- (1 vs 4)
- (2 vs 3)
- (2 vs 4)
- (3 vs 4)

Each classifier only uses **data from the two classes** being compared.

◆ Training Process

For each pair (i, j):

- Keep only samples of classes i and j
- Train a binary SVM to separate these two

This reduces complexity because each model is trained on **smaller datasets**.

◆ Prediction

When a test sample arrives:

- Each classifier votes for one of the two classes
- All votes are collected
- The class with **maximum votes** wins

This is called **majority voting**.

Example outcome for a 4-class problem:

Classifier	Winner
1 vs 2	1
1 vs 3	3
1 vs 4	1
2 vs 3	2
2 vs 4	4
3 vs 4	3

Votes:

- 1 → 2 votes
- 2 → 1 vote
- 3 → 2 votes
- 4 → 1 vote

If a tie occurs, usually:

- Choose the class with the highest decision value
- or run tie-breaking heuristics

◆ Advantages of OvO

- ✓ Works extremely well in practice
- ✓ Each classifier sees **only two classes** → **smaller, simpler datasets**
- ✓ Faster training when each class has many samples
- ✓ More accurate than OvR for many problems
- ✓ No class imbalance issue

OvO is the default strategy in popular libraries like **LIBSVM** and often in **scikit-learn** for SVM.

◆ Disadvantages of OvO

- ✗ Need many classifiers → $O(K^2)$
- ✗ Prediction requires running all classifiers → slower for large K
- ✗ Voting may produce ties (rare but possible)

Still, for most practical K (3–10 classes), OvO is usually superior.

★ 9.3 OvR vs OvO — Comparison Table

Feature	OvR	OvO
# of models	K	$K(K-1)/2$
Training set per model	All data	Only 2 classes
Training speed	Faster for small datasets	Faster for large datasets
Prediction speed	Faster	Slower (more models)
Accuracy	Moderate	Higher in many cases
Class imbalance	Problematic	No issues
Implementation	Simple	More complex

★ 9.4 Which One Should You Use?

✓ OvO is preferred when:

- You want higher accuracy
- You have moderate number of classes (3–20)
- Libraries use optimized pairwise SVMs

✓ OvR is preferred when:

- Huge number of classes (e.g., 100–1000)
 - Speed is important
 - You want a simpler implementation
-

★ 9.5 Key Takeaways

- SVM is fundamentally binary → multi-class is built using strategies.
 - **One-vs-Rest (OvR)**: Train K models; each class is vs the rest.
 - **One-vs-One (OvO)**: Train $K(K-1)/2$ models; each class is compared pairwise.
 - OvO generally yields higher accuracy but requires more models.
 - OvR is simpler and more efficient for very large K .
 - Modern SVM implementations often default to **One-vs-One** due to high accuracy and robust performance.
-

★ 10. Symbolic Kernels and Regression

with focus on:

- How SVMs extend to **regression**
 - What **Support Vector Regressor (SVR)**?
 - How kernels power non-linear regression
 - Why does SVR behave differently from classical regression
-

★ 10.1 Extending SVMs to Regression Tasks

SVMs are originally designed for **classification**, where the goal is to find a margin that separates classes.

But the powerful ideas behind SVMs—margins, support vectors, kernels—can also be applied to **predict continuous values** instead of classes.

This leads to **Support Vector Regression (SVR)**.

✓ In classification:

Find a hyperplane that separates classes with maximum margin.

✓ In regression:

Find a function (line, curve, surface) that:

- Fits the data
- Is as **flat** (simple) as possible
- Ignores small errors
- Penalizes only **large** deviations

This combination of:

- simplicity
- margin ideas
- kernels

gives SVM-based regression very powerful generalization abilities.

★ 10.2 What Is Support Vector Regression (SVR)?

SVR is the regression version of SVM.

Instead of separating classes, SVR tries to fit a **smooth function** to data.

✓ Goal of SVR:

Find a function ($f(x)$) that deviates from the true values (y_i) by **at most ϵ** , and is as flat (low-weight) as possible.

Here:

- ϵ (epsilon) defines an **error tolerance zone**
- Only points **outside** this ϵ -tube are penalized
- These points become **support vectors**

★ 10.3 ϵ -Insensitive Loss Function

SVR introduces a special loss:

$$L_{\epsilon}(y, f(x)) = \begin{cases} 0 & , \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & , \text{otherwise} \end{cases}$$

This means:

- ✓ Small errors (inside tube) → **ignored**
- ✓ Only large errors (outside tube) → **penalized**

This is VERY different from ordinary least squares regression (which punishes all errors).

★ 10.4 Regression Tube (ϵ -Tube)

SVR constructs a "tube" around the regression line:

$$\begin{array}{c} | y + \epsilon \\ | \\ ===== | ===== f(x) \text{ curve} ===== \\ | \\ | y - \epsilon \end{array}$$

Points inside the tube → perfect

Points outside the tube → violations → support vectors

This helps:

- Ignore noise

- Reduce overfitting
 - Focus on important deviations
-

★ 10.5 Optimization Objective of SVR

SVR solves:

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

Subject to:

$$y_i - w^T x_i - b \leq \epsilon + \xi_i$$

$$w^T x_i + b - y_i \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

Where:

- (ξ_i, ξ_i^*) measure error **outside** the ϵ -tube
 - (C) controls the penalty for errors, similar to soft-margin SVM
 - The goal is still to keep $\|w\|$ small \rightarrow smooth function
-

★ 10.6 Support Vectors in Regression

In SVR:

- Support vectors are **points outside the ϵ -tube**
- Points inside the tube don't contribute to the model
- SVR prediction depends **ONLY** on support vectors:

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

Thus:

SVR ignores easy examples.

Only difficult points influence the regression curve.

This makes SVR **sparse**, **robust**, and **generalizes well**.

★ 10.7 Role of Kernels in SVR (Symbolic Kernels)

Just like SVMs for classification, SVR also uses kernels.

✓ Why kernels matter?

- Many regression problems are **non-linear**
- Kernels map data to a **higher-dimensional space**
- In that space, a simple linear function behaves like a **complex curve** in the original space

Common kernels used in SVR:

Kernel	Effect in Regression
Linear	Linear regression with margin ideas
Polynomial	Curves of increasing complexity
RBF (Gaussian)	Highly smooth non-linear regression; most popular
Sigmoid	Neural-network-like behavior

★ 10.8 Symbolic Kernels

The professor's slide mentions "Symbolic kernels".

This refers to the fact that:

- ✓ Kernels are not numeric transformations
- ✓ They are **symbolic functions** that compute similarity
- ✓ They allow infinite-dimensional mappings symbolically

For example:

- Polynomial kernel symbolically creates all polynomial combinations
- RBF kernel symbolically maps to infinite dimensions

- No explicit feature transformation is computed

This is especially powerful in regression when the relationship is:

- curved
- non-linear
- complex
- smooth
- multi-dimensional

The kernel handles the complexity **symbolically**, while SVR focuses on fitting a smooth curve.

★ 10.9 Why Choose SVR Over Standard Regression?

- ✓ Handles non-linear data naturally (via kernels)
- ✓ Robust to outliers (ϵ -insensitive loss)
- ✓ Produces smooth functions
- ✓ Sparse model (only support vectors matter)
- ✓ Strong generalization (margin-based learning)
- ✓ Works well even with limited data

These advantages make SVR extremely popular in:

- Finance (stock prediction)
 - Engineering
 - Time series
 - Bioinformatics
 - Signal/noise modeling
-

★ 10.10 Final Summary (Exam-Ready)

- SVM extends to regression through **Support Vector Regression (SVR)**.

- SVR uses an **ϵ -insensitive loss**, ignoring small errors.
 - Only points that violate the ϵ -tube become **support vectors**.
 - SVR minimizes **model complexity + large deviations**.
 - Kernels allow SVR to perform **non-linear regression** efficiently.
 - Symbolic kernels mean transformations are handled implicitly.
 - SVR produces smooth, robust, generalizable regression curves.
-

★ 11. Final Summary of SVM Concepts

This final summary ties together everything about Support Vector Machines (both classification and regression). It highlights what makes SVMs unique, powerful, and fundamentally different from other machine learning algorithms like the Perceptron and Neural Networks.

★ 11.1 SV-Classifiers and SV-Machines Overview

✓ SV-Classifiers (SVC)

These are **Support Vector Classifiers**, i.e., SVMs used for **classification** tasks.

They include:

- **Hard-Margin SVM** (strict separation)
- **Soft-Margin SVM** (allows some misclassification)
- **Kernel SVM** (handles non-linear boundaries using kernel functions)

They produce:

- **Maximum-margin decision boundaries**
 - **Sparse solutions** (use only support vectors)
 - **High generalization ability**
-

✓ SV-Machines (SVM in general)

"SVM" is the broader umbrella term that includes:

- **Support Vector Classifier (SVC)** for classification
- **Support Vector Regressor (SVR)** for regression

The core philosophy of SVMs:

Use margins, support vectors, and kernels to build simple, generalizable models, even in high-dimensional spaces.

SVMs aim to find:

- A **hyperplane** (linear SVM)
 - Or a **hypersurface** (kernel SVM)
- that best separates or fits the data.

★ 11.2 Unique and Cushioned Separators

This phrase refers to the **margin** of SVMs.

✓ **Unique separator**

The SVM solution is **unique** because:

- SVM optimization is a **convex** quadratic programming problem
- Convex problems have **unique global optima**
- The margin-maximizing hyperplane exists uniquely (no multiple local minima)

Unlike many ML models, SVMs don't depend on:

- random initialization
- stochastic training
- multiple local minima

→ The solution is deterministic.

✓ **Cushioned separator**

The “cushion” refers to the **margin**—the buffer zone around the decision boundary.

SVMs don’t just separate the classes; they create a **thick margin** on both sides of the hyperplane:

Class A | ← margin → | Decision | ← margin → | Class B

This “cushion” provides:

- **Better generalization**
- **Noise tolerance**
- **Stability**
- **Less overfitting**

Only the closest points (support vectors) determine the boundary—the other data points do **not** influence the model.

This is very different from other classifiers.

★ 11.3 Comparison with Perceptron (PLA)

✓ Perceptron Learning Algorithm (PLA)

PLA:

- Works only if data is **perfectly linearly separable**
- Fails to converge otherwise
- Is highly sensitive to noisy or overlapping classes
- Has no concept of **margin**, **slack**, or **kernels**

✓ SVM vs PLA

Aspect	Perceptron	SVM
Handling of non-linearity	✗ No	✓ Yes (kernels)
Handling of overlapping or noisy data	✗ Poor	✓ Soft margin + slack variables

Aspect	Perceptron	SVM
Margin	✗ None	✓ Maximum-margin separator
Convergence guarantee	✗ Only if data is separable	✓ Always (convex optimization)
Generalization	Moderate	Excellent

Key point:

SVM is essentially a **smarter, more stable, margin-based version of PLA**, with kernel extensions.

★ 11.4 Comparison with Neural Networks

Neural Networks are highly flexible, but they come with issues:

✓ Problems with Neural Networks:

- Multiple local minima
- Sensitive to initialization
- Risk of overfitting without regularization
- Require large datasets
- Require careful tuning (learning rate, layers, optimizers, etc.)
- Harder to interpret

✓ Advantages of SVMs:

1. Convex optimization

- Guarantees a **global optimum**
- More stable and predictable

2. Margin maximization

- Better generalization
- Less overfitting on small/medium datasets

3. Kernel trick

- Handles complex non-linear boundaries
- Without needing deep architectures

4. Works well with limited data

- Neural networks need large data
- SVMs shine with small to medium datasets

5. Sparse model

- Only support vectors matter
- Avoids unnecessary complexity

Conclusion:

Neural Networks are more flexible, but SVMs are more generalized, stable, mathematically grounded, and robust with small datasets.

This is why SVMs were hugely popular before deep learning took over for massive datasets.

★ 11.5 Regression Capability of SVMs (SVR)

SVMs are not limited to classification—they can also perform **regression** via **Support Vector Regression (SVR)**.

✓ Key differences from classical regression:

SVR:

- Uses an **ϵ -insensitive loss function**
- Ignores small errors
- Only penalizes deviations outside an ϵ -tube
- Produces **smooth, flat** functions (similar to margin maximization)
- Uses **support vectors** for prediction
- Can become **non-linear** using kernels (e.g., RBF)

This allows SVR to:

- Handle noise
 - Avoid overfitting
 - Build smooth, stable regression curves
 - Model non-linear relationships efficiently
-

★ 11.6 Big Picture: What Makes SVMs Special

- ✓ Maximum-margin principle
 - strong generalization
 - ✓ Sparse solution
 - only support vectors matter
 - ✓ Kernel trick
 - solve non-linear problems efficiently
 - ✓ Convex optimization
 - single global optimum
 - ✓ Works for both classification and regression

 - ✓ Best suited for:
 - Medium-sized datasets
 - High-dimensional data (text, bioinformatics, finance)
 - Problems with non-linear but smooth decision boundaries
-

★ Final Takeaway (Memory Shortcut)

SVM =

Margin + Support Vectors + Kernels + Convex Optimization

→ **Stable, generalizable, non-linear**, and works for both **classification & regression**.

★ 12. Which Method Is Most Suitable for Non-Separable, Non-Linear Data?

Which of the following is most suitable for classifying non-separable data having a non-linear decision boundary?

1. Perceptron Learning (Pocket algorithm)
2. Naïve Bayes
3. Support Vector Machines
4. Neural Networks

Let's analyze each option deeply.

★ Option 1: Perceptron / Pocket Algorithm

PLA or Pocket algorithm:

- Can only handle **linearly separable data**
- Cannot learn curved or non-linear boundaries
- Will not converge if data overlaps or is noisy
- No concept of kernels or feature mapping

Pocket algorithm improves over PLA by saving the best solution so far, but **still cannot handle non-linear structure**.

✗ **Not suitable for non-linear decision boundaries.**

★ Option 2: Naïve Bayes

- A simple, probabilistic classifier

- Assumes **conditional independence**
- Forms decision boundaries based on Gaussian/Multinomial distributions
- Usually produces **linear** or simple boundaries
- Not powerful enough for complex non-linear separations
- Sensitive to incorrect independence assumptions

✗ Not suitable for complex non-linear structures.

★ **Option 3: Support Vector Machines (SVM)**

This is the best choice because:

✓ **Can handle non-separable data**

Using **soft-margin SVM**, misclassifications and margin violations are allowed.

✓ **Can handle non-linear decision boundaries**

Using **kernel trick**, SVM projects data into a high/infinite-dimensional feature space.

Example kernels:

- Polynomial
- Gaussian RBF
- Sigmoid

✓ **Finds smooth, optimal boundaries**

The **maximum-margin principle** ensures:

- Good generalization
- Robustness to noise
- Stable performance

✓ **Uses only support vectors**

Sparse, efficient, and powerful.

✓ Solves a convex optimization problem

Guarantees a global optimum.

★ Final Verdict:

SVM is the most suitable for non-separable, non-linear classification.

★ Option 4: Neural Networks

Neural Networks *can* handle non-linear boundaries, sometimes better than SVMs, but:

! Weaknesses in comparison to SVM:

- Require large amounts of data
- Require tuning of many hyperparameters
- Multiple local minima
- Sensitive to initialization
- Risk of overfitting
- Complex to train

While neural nets are very powerful, they're **not always the best choice** for small-to-medium datasets or when interpretability and stability are important.

✓ They can handle non-linearity,

✗ but SVM is generally *more reliable and stable* for these kinds of problems.

★ Correct Answer: 3. Support Vector Machines

Because SVM + kernels is specifically designed for:

- non-linear separation
- non-separable datasets
- noisy data

- high-dimensional input

It's the best match for the scenario provided.

★ Multi-Class Recap

✓ **One-vs-Rest (OvR)**

Train **K models** for K classes.

Each model: class k vs. all others.

✓ **One-vs-One (OvO)**

Train **$K(K-1)/2$ models** for all pairs.

Prediction uses majority vote.

Most practical SVM libraries (like LIBSVM, sklearn's SVC) use **One-vs-One** because it performs better.

★ Final Wrapped Summary of SVM Concepts

- **SVC** handles classification; **SVR** handles regression.
- SVM uses **margin maximization** → yields cushioned, stable separations.
- **Soft-margin SVM** handles non-linearly separable data.
- **Kernel SVM** handles non-linear boundaries efficiently.
- **Dual formulation** allows kernels and uses support vectors.
- SVMs outperform PLA for non-separable data.
- SVMs are more stable and generalizable than Neural Networks for small datasets.
- SVMs also perform regression via **Support Vector Regression (SVR)**.
- Multi-class classification uses **OvR** or **OvO** strategies.

- Best algorithm for non-linear, non-separable data → **SVM (with kernels)**.
-

SVM – Kernel Property

★ 1. If $(k_1(x, x'))$ is a valid kernel → it corresponds to a dot product

A kernel is **valid** if it can be written as an inner product in some (possibly infinite-dimensional) feature space:

$$k_1(x, x') = \phi(x)^T \phi(x')$$

This is **Mercer's condition**:

| A valid kernel must be equal to a dot product of some feature transform.

This transform $(\phi(x))$ may be high-dimensional or infinite-dimensional — but it **exists**.

★ 2. Scaling a valid kernel by a positive constant is still valid

If:

$$k_1(x, x') = \phi(x)^T \phi(x')$$

then:

$$c \cdot k_1(x, x') = (c^{1/2} \phi(x))^T (c^{1/2} \phi(x'))$$

This is still a **dot product**, so still a valid kernel.

The slide refers to:

- $(u(x) = \frac{1}{c} \phi(x))$
- So $(ck_1(x, x') = u(x)^T u(x'))$

This shows that **scalar multiples preserve kernel validity**.

★ 3. Goal of the slide

We want to check whether:

$$k(x, x') = \exp(-\gamma|x - x'|) \quad \text{or} \quad \exp(-\gamma|x - x'|^2)$$

is a **valid kernel**.

The slide uses the squared version in the approach, which is the classical **Gaussian RBF kernel**:

$$k(x, x') = \exp(-\gamma|x - x'|^2)$$

We must show it equals an inner product:

$$k(x, x') = \Phi(x)^T \Phi(x')$$

★ 4. Approach the professor suggests

The slide says:

Split the exponential term to show

$$k(x, x') = f(x), k_2(x, x'), f(x')$$

Meaning we want to express the kernel as:

something depending only on x \times a kernel in (x, x') \times
something depending only on x'

This is important because:

- If $(k_2(x, x'))$ is a valid kernel
 - And $(f(x)f(x'))$ multiplies it
- The whole expression is still a **valid kernel**

This is a known closure property of kernels.

★ 5. Breaking the RBF kernel mathematically

Start with:

$$\|x - x'\|^2 = \|x\|^2 + \|x'\|^2 - 2x^T x'$$

So:

$$e^{-\gamma\|x-x'\|^2} = e^{-\gamma\|x\|^2} e^{-\gamma\|x'\|^2} e^{2\gamma x^T x'}$$

Let:

- $(f(x) = e^{-\gamma\|x\|^2})$
- $(f(x') = e^{-\gamma\|x'\|^2})$
- $(k_2(x, x') = e^{2\gamma x^T x'})$

Then the kernel becomes:

$$k(x, x') = f(x) \cdot k_2(x, x') \cdot f(x')$$

This is exactly what the slide suggests.

★ 6. Why is this decomposition useful?

Because we know:

✓ **If $(k_2(x, x') = e^{2\gamma x^T x'})$, then it is a valid kernel**

Proof idea:

$$e^{2\gamma x^T x'} = \sum_{n=0}^{\infty} \frac{(2\gamma x^T x')^n}{n!}$$

Each term $((x^T x')^n)$ is a **polynomial kernel**, and:

- Non-negative sum
- Of valid kernels
- With positive coefficients

→ still a **valid kernel**.

✓ **Multiplying by $(f(x)f(x'))$ keeps it valid**

Kernel closure property:

$$k(x, x') = f(x)k_2(x, x')f(x')$$

is a valid kernel if:

- (k_2) is valid
- $(f(x) \geq 0)$

Both conditions hold.

Thus:

$$k(x, x') = \exp(-\gamma|x - x'|^2)$$

is a **valid kernel**.

★ 7. Intuitive Explanation

The Gaussian RBF kernel represents an **inner product in an infinite-dimensional space**:

- It maps each point into infinitely many dimensions (basis functions)
- The similarity decays with squared distance
- This infinite feature expansion can be shown mathematically via series expansion

The slide's approach shows this **constructively**, step-by-step.

★ 8. Final Takeaway

- ✓ A kernel is valid if it corresponds to an inner product in some feature space
- ✓ Scaling or transforming kernels preserves validity
- ✓ Gaussian RBF kernel can be decomposed into known valid kernels
- ✓ Multiplying valid kernels with positive functions preserves validity
- ✓ Therefore, **RBF kernel is a valid kernel**

This is a classic demonstration used in machine learning theory to justify using the Gaussian RBF kernel in SVM.

SVM/SVR for Time-Series

★ 1. Why Use SVM/SVR for Time-Series?

Time-series data often show:

- **Non-linear patterns**
- High noise
- Small sample sizes
- Seasonal/periodic behavior
- Sudden changes

Traditional linear methods may struggle here.

SVR is powerful because it:

- Models **non-linear relationships** through kernels
- Controls noise using an ϵ -insensitive loss
- Avoids overfitting using margin maximization
- Performs well even with **small datasets**
- Handles multi-dimensional input (lags, exogenous variables)

This makes SVR a strong candidate for forecasting tasks like stock prices, weather, load/power demand, traffic, and sales.

★ 2. How SVM Handles Time-Series: The Core Idea

SVM **does not forecast directly** like ARIMA.

Instead, we **convert the time-series into a supervised learning dataset** using **lag features**.

For example, given a sequence:

$$x_{t-3}, x_{t-2}, x_{t-1}, x_t$$

We create supervised pairs like:

$$\text{Input: } (x_{t-3}, x_{t-2}, x_{t-1}) \rightarrow \text{Target: } x_t$$

This is called **sliding-window** or **lag-based feature creation**.

★ 3. Steps to Perform Time-Series Forecasting Using SVR

✓ Step 1: Convert time-series to supervised learning form

Example for forecasting next day's price:

- Input features = values at days (t-1, t-2, ..., t-p)
- Output = value at day (t)

$$X_t = [x_{t-1}, x_{t-2}, \dots, x_{t-p}]$$

$$y_t = x_t$$

This converts your time series into a matrix suitable for SVR.

✓ Step 2: Choose Kernel Function

Common kernels:

🟦 RBF (Gaussian) Kernel → Most popular

Captures non-linear dependencies extremely well.

🟡 Polynomial Kernel

Useful when the relationship is moderately non-linear.

🔴 Linear Kernel

Used if the data is almost linear.

RBF kernel is **usually best** for time-series.

✓ Step 3: Train SVR Model

Train using:

- Kernel = RBF
- Hyperparameters: C, ϵ , γ

Hyperparameters need tuning (grid search or validation).

✓ Step 4: Forecast Future Values

Two approaches:

1 One-step ahead prediction

Predict (x_{t+1}) from past lag values.

2 Multi-step forecast

Use predicted values as inputs for further predictions (recursive).

★ 4. Why SVR Works Well for Time-Series

✓ Handles Non-linearity

Time-series often have:

- Non-linear patterns
- Volatility
- Threshold effects

SVR with RBF kernel captures this easily.

✓ Robust to Noise

SVR's ϵ -insensitive tube ignores small fluctuations:

- Minor random movements → ignored
- Only big deviations → matter

This is great for noisy data like finance.

✓ Avoids Overfitting

Margin maximization ensures generalization, even with many features.

✓ Works on Small Datasets

Unlike neural networks, which need large datasets, SVR works well even with:

- 100 samples
 - limited historical data
-

★ 5. Example Use Cases of SVM in Time-Series

- Stock price prediction
- Weather forecasting
- Traffic volume forecasting
- Sales forecasting
- Energy load prediction
- Sensor signal prediction
- Demand forecasting

SVR is especially useful in domains where:

- Non-linearity exists
 - Data is limited
 - Noisy observations are present
-

★ 6. Important Hyperparameters in SVR for Time-Series

✓ **C (Cost/Penalty)**

Controls model complexity

Large C → more flexible, risk of overfitting

Small C → simpler, may underfit

✓ **ϵ (Epsilon)**

Width of the tube

Large ϵ → ignores more errors

Small ϵ → more sensitive to noise

✓ **γ (Gamma)**

Controls kernel spread

High gamma → overfit

Low gamma → underfit

Hyperparameter tuning is critical.

★ 7. SVR vs LSTM / Neural Networks for Time-Series

Feature	SVR	LSTM / Neural Networks
Works with small dataset	★ Excellent	✗ Needs large data
Handles non-linearity	★ Yes (via kernels)	★ Yes
Training time	Fast	Slow
Risk of overfitting	Low	High
Handles long-term dependencies	✗ No	★ Yes
Interpretability	Medium	Low
Easy to tune	Medium	Hard

Conclusion:

- SVR is ideal for **short-term forecasting** with **small to medium-sized data**.
- LSTMs/transformers dominate when **long sequences** and **large datasets** are available.

★ 8. Visual Intuition of SVR in Time-Series

Imagine a noisy curve. SVR creates a **smooth function** that fits inside an ϵ -tube:

Real signal: ~~~~~~
SVR prediction: ----- smooth curve -----

- Small wiggles are ignored
- Large shifts create support vectors
- The boundary follows the underlying trend

Perfect for forecasting.

★ 9. When Should You Use SVM for Time-Series?

Use SVR when:

- You have limited data (<5000 samples)
- Patterns are non-linear
- Data is noisy
- You want a stable, robust model with little tuning
- You need explainability

Do NOT use SVR when:

- You have very long sequences (LSTMs are better)
 - You need probabilistic forecasts
 - You require interpretable trend/seasonality decomposition
-

★ Final Summary

Time-series analysis with SVM (via SVR) involves:

- Converting the sequence into supervised learning with lag features
- Training SVR with a kernel (usually RBF)
- Using ϵ -insensitive loss for noise robustness
- Predicting future values using sliding windows

SVR is:

- Non-linear
- Stable
- Generalizable
- Excellent for small, noisy datasets

It remains one of the **most effective machine learning forecasting tools**, especially when deep learning is unnecessary or overkill.
