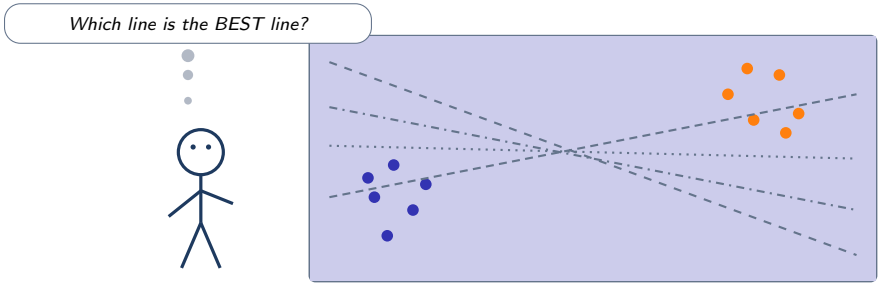
SVM: A Simple Lecture with All Formulas

Maximum Margin, Kernel Trick, and Worked Examples – BSc Data Science

60 Minutes



Four candidate lines all separate the data. Which one will the SVM pick?

What Is Classification?

Plain English: Given examples of two groups, predict which group a new example belongs to. That is it.

Everyday analogies.

- Sorting incoming email into *spam* vs *ham*
- Flagging loan applications as *default* vs *repay*
- Marking transactions as *fraud* vs *legitimate*

The common shape.

- Input: a **vector x** — an ordered list of numbers (one per **feature**)
- **Feature**: a measurable property, one column in the data table
- Output: a **class label** — the category we're predicting ($+1$ or -1)
- Rule: learned from past labelled examples

Why a binary rule?

Binary = two classes only, e.g. $\{+1, -1\}$. In this lecture we focus on two classes to keep the geometry simple.

Multi-class extension. Run one binary SVM per class ("one-vs-rest") or per pair ("one-vs-one"), then vote. The core algorithm stays binary.

Goal of this deck: by the end you can compute the separating line by hand and evaluate three kernels on a single test point.

Classification = input a vector, output a label. SVM is one specific method for learning the rule.

What Is a Vector? Quick Refresher

Definition. A **vector** is an *ordered list of numbers*. Each entry corresponds to one feature of the example.

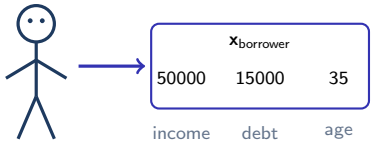
Borrower example (3 features).

$$\mathbf{x}_{\text{borrower}} = (\underbrace{50000}_{\text{income}}, \underbrace{15000}_{\text{debt}}, \underbrace{35}_{\text{age}}).$$

Email example (3 features).

$$\mathbf{x}_{\text{email}} = (\underbrace{7}_{\#\text{free}}, \underbrace{0}_{\#\text{meeting}}, \underbrace{4}_{\#\text{viagra}}).$$

Take-away. A vector is just a row of the data table, minus the label.



Geometry. In 2D we can draw a vector as an arrow from the origin. In higher dimensions (3, 10, 10000), the picture abstract — but the arithmetic is the same.

Vector = ordered list of numbers. One entry per feature. That is all a vector is.

One-vs-Rest & One-vs-One Explained

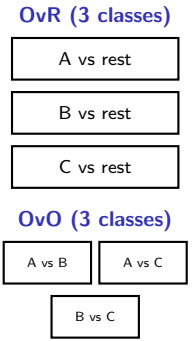
The binary core. SVM is natively a 2-class method. For $K > 2$ classes we build several binary SVMs.

One-vs-Rest (OvR). Train K **binary SVMs**: class k vs everyone else.

- 3 classes $\Rightarrow$ 3 **binary problems**
- 5 classes $\Rightarrow$ 5 **binary problems**
- At prediction: pick the class whose SVM score is highest

One-vs-One (OvO). Train one **binary SVM** for every *pair*.

- 3 classes $\Rightarrow$ 3 pairs (binary SVMs)
- 5 classes $\Rightarrow$ $\binom{5}{2} = 10$ binary SVMs
- At prediction: each pair-SVM votes; majority wins



In scikit-learn. `SVC` defaults to OvO. `LinearSVC` defaults to OvR.

Multi-class via many 2-class models. OvR: K SVMs. OvO: $\binom{K}{2}$ SVMs.

Three Finance-Flavoured Examples

Credit default prediction.

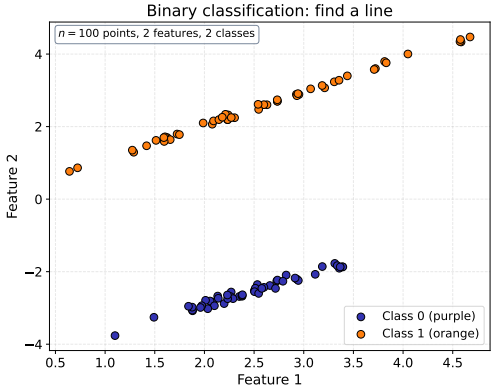
- Features: income, debt ratio, credit history score
- Classes: $+1$ repay on time, -1 default

Spam detection.

- Features: word counts, sender reputation
- Classes: $+1$ legitimate, -1 spam

Fraud flagging.

- Features: transaction amount, merchant category, time of day
- Classes: $+1$ normal, -1 fraud



In every case: a rule maps features to one of two labels. SVM learns that rule from examples.

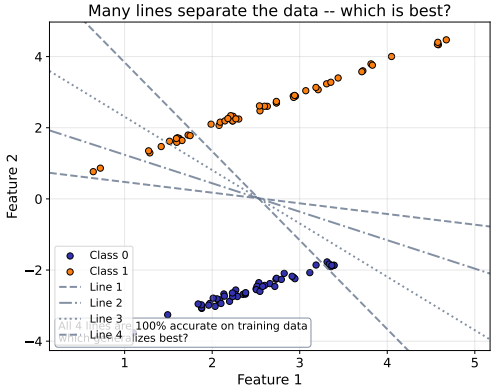
Many Lines Can Separate the Data

The embarrassment of riches. For a cleanly separable dataset, **infinitely many** straight lines achieve 100% training accuracy.

The four lines on the right all score perfect on the training data. But they are *not* equally good on new data.

Question to the reader.

- Which of the four will best handle a slightly noisier future point?
- Which one sits "farthest" from every training example?
- **How do we even define "best"?**



Training accuracy cannot discriminate between these four lines. We need an additional principle.

Why Is Picking the "Best" Line Hard?

The training data cannot tell us.

- All four lines fit the training data *perfectly*
- Their training loss is identical: zero
- So the loss function alone cannot choose between them

We need a PRINCIPLE to pick. The principle SVM chooses:

"Pick the line that is as far as possible from both classes."

That single idea — maximum *margin* — is the whole SVM story. The rest of this deck fills in the maths.

Alternative principles other methods use.

- **Logistic regression** picks the line that maximises data likelihood under a sigmoid
- **Perceptron** picks any line that gets zero training error
- **Decision tree** does not use lines at all

SVM is the method whose principle is "stay as far as possible from every training point".

Zero training error is not enough. SVM adds the rule: maximise the distance to the closest points.

The Decision Function (Formula F1)

Formula F1.

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

Plain English.

- Compute a number: $\mathbf{w} \cdot \mathbf{x} + b$
- If the number is *positive*, predict $+1$
- If the number is *negative*, predict -1
- If the number is exactly zero, the point sits on the decision boundary

That is it. No non-linearity, no magic — just a weighted score followed by its sign.

What $\mathbf{w} \cdot \mathbf{x}$ means concretely.

If $\mathbf{w} = (w_1, w_2)$ and $\mathbf{x} = (x_1, x_2)$, then the **dot product** is the *sum of products* of matching entries — a single scalar:

$$\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + w_2 x_2$$

Geometric reading.

- $\mathbf{w}$ is a vector pointing perpendicular to the **decision boundary** (the flat surface, i.e. hyperplane, where $f(\mathbf{x}) = 0$)
- b shifts the boundary toward or away from the origin
- The boundary is the set where $\mathbf{w} \cdot \mathbf{x} + b = 0$

F1 is one line of code: predict sign of a weighted sum. The whole game is choosing $\mathbf{w}$ and b.

What Is a Dot Product?

Definition. The **dot product** of two vectors $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$ is a single number, the *sum of products* of matching entries:

$$\mathbf{w} \cdot \mathbf{x} = \underbrace{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}_{\text{sum of matched products}}$$

Tiny worked example. $\mathbf{w} = (1, 2)$, $\mathbf{x} = (3, 4)$:

$$\mathbf{w} \cdot \mathbf{x} = 1 \cdot 3 + 2 \cdot 4 = 3 + 8 = \mathbf{11}.$$

Key properties.

- Result is one number (not a vector)
- Commutative: $\mathbf{w} \cdot \mathbf{x} = \mathbf{x} \cdot \mathbf{w}$
- Distributive: $\mathbf{w} \cdot (\mathbf{x} + \mathbf{y}) = \mathbf{w} \cdot \mathbf{x} + \mathbf{w} \cdot \mathbf{y}$

Geometric meaning.

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \, \|\mathbf{x}\| \cos(\theta)$$

where θ is the angle between the two arrows.

- Same direction ($\theta = 0°$): $\cos\theta = 1$, dot product is largest
- **Perpendicular** (**orthogonal**, $\theta = 90°$, a right angle): $\cos\theta = 0$, dot product is zero
- Opposite direction ($\theta = 180°$): $\cos\theta = -1$, dot product is most negative

That is why SVM uses it. The score $\mathbf{w} \cdot \mathbf{x} + b$ measures alignment of $\mathbf{x}$ with $\mathbf{w}$, shifted by b.

Dot product = sum of products of matched entries. Geometry: $\|\mathbf{w}\| \|\mathbf{x}\| \cos\theta$.

What Is **w**? What Is b?

w is a DIRECTION.

- A vector in the same space as **x**
- Perpendicular to the decision boundary
- Points from the "-1 side" toward the "$+1$ side"

b is a THRESHOLD.

- A scalar number
- Shifts the boundary away from the origin
- Without b, the boundary would always pass through $(0, 0)$

Tiny 2D example. $\mathbf{w} = (1, 1)$, $b = -1$. Boundary: $x_1 + x_2 = 1$.

At $(2, 2)$: $1 \cdot 2 + 1 \cdot 2 - 1 = 3 > 0 \Rightarrow +1$.
At $(0, 0)$: $1 \cdot 0 + 1 \cdot 0 - 1 = -1 < 0 \Rightarrow -1$.

Why perpendicular?

The set $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$ is a line (in 2D) or **hyperplane** — a flat surface, the generalisation of a line (or plane) to any dimension — in higher dim.

From vector geometry, the **gradient** (the vector of partial derivatives, always **perpendicular** i.e. at $90°$/right angle to level sets) of $\mathbf{w} \cdot \mathbf{x} + b$ with respect to **x** is just **w**. This is a slope read as a direction. So **w** is perpendicular to the boundary.

Mental picture. **w** is an arrow *sticking out* of the boundary pointing toward the $+1$ class.

w = direction perpendicular to the boundary. b = how far the boundary sits from the origin.

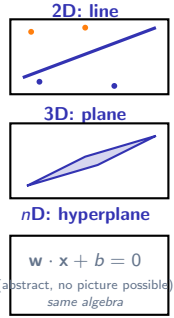
What's a Hyperplane? Decision Boundary Defined

Hyperplane = a *flat surface*, the generalisation of a line, one dimension lower than the space it lives in.

Space	Hyperplane	Equation
2D	line	$w_1x_1 + w_2x_2 + b = 0$
3D	plane	$w_1x_1 + w_2x_2 + w_3x_3 + b = 0$
nD	hyperplane	$\mathbf{w} \cdot \mathbf{x} + b = 0$

Decision boundary = the hyperplane where $f(\mathbf{x}) = 0$; one side is class $+1$, other side is class -1.

Tiny check. Take $\mathbf{w} = (1, 1), b = 0$. The boundary is $x_1 + x_2 = 0$. Points $(1, -1)$ and $(-1, 1)$ sit *on* this boundary: $1 + (-1) = 0$, $(-1) + 1 = 0$. ✓



Hyperplane = flat surface, one dim lower than its space. Decision boundary is where $f(\mathbf{x}) = 0$.

Worked Example 1: Setup

Three data points. That is all.

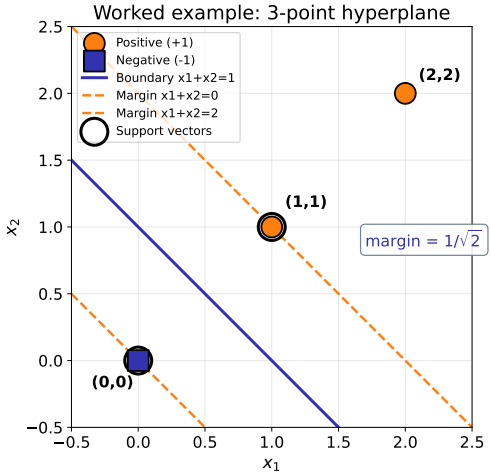
- Positives: $(1,1)$, $(2,2)$ $\quad [y=+1]$
- Negative: $(0,0)$ $\quad [y=-1]$

Goal. Find the max-margin separating line by hand.

Preview of the answer. By symmetry, $\mathbf{w}$ will point along the direction $(1,1)$. We will discover

$$\mathbf{w} = (1,1), \quad b = -1, \quad \text{margin} = \frac{1}{\sqrt{2}}.$$

All arithmetic will be shown over the next five slides.



Three points. By the end of this example we know w, b, and the margin, with no optimisation solver.

The Margin Idea

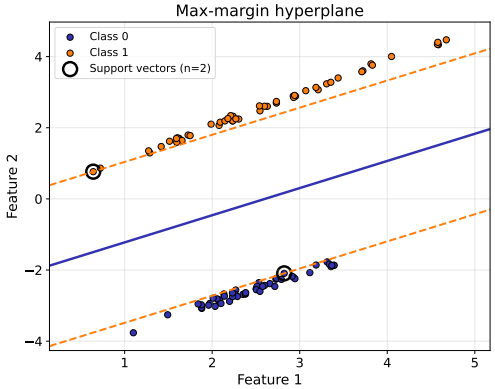
Definition (informal).

Margin = distance from the boundary to the CLOSEST training point.

Why larger is better.

- Future test points will usually sit near training points but slightly displaced
- A wider margin means small displacements do not cross the boundary
- More robust $\Rightarrow$ better generalisation

SVM's rule. Among all lines that separate the classes, pick the one with the *largest* margin.



Margin = "safety buffer" around the boundary. SVM maximises it over all separating lines.

Computing Distances: 2D Geometry Recap

Distance from a point to a line.

For a line $w_1x + w_2y + b = 0$ in 2D and a point (a, b_0) (not to be confused with the line's b):

$$\text{dist} = \frac{|w_1a + w_2b_0 + b|}{\sqrt{w_1^2 + w_2^2}} = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|}$$

Why the denominator? $\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2 + \dots}$ is the **length of w** (Euclidean norm — Pythagoras in n dimensions). Because $\mathbf{w} \cdot \mathbf{x} + b$ scales with $\|\mathbf{w}\|$, we want a *geometric* distance, so we divide by $\|\mathbf{w}\|$.

Why the absolute value? Distance is non-negative. Points on one side give positive scores, the other side negative.

Tiny sanity check.

Line $x_1 + x_2 - 1 = 0$, so $\mathbf{w} = (1, 1)$ and $b = -1$. $\|\mathbf{w}\| = \sqrt{1^2 + 1^2} = \sqrt{2}$.

Distance from $(0, 0)$:

$$\frac{|1 \cdot 0 + 1 \cdot 0 - 1|}{\sqrt{2}} = \frac{1}{\sqrt{2}} \approx 0.707.$$

Distance from $(1, 1)$:

$$\frac{|1 \cdot 1 + 1 \cdot 1 - 1|}{\sqrt{2}} = \frac{1}{\sqrt{2}} \approx 0.707.$$

Both distances equal $1/\sqrt{2}$. Same side? Opposite sides? Sign of $\mathbf{w} \cdot \mathbf{x} + b$ tells us.

Distance from point to line = $|\mathbf{w} \cdot \mathbf{x} + b| / \|\mathbf{w}\|$. Keep this formula in mind.

What Is $\|\mathbf{w}\|$ (the Norm)?

Definition. The **Euclidean norm** $\|\mathbf{w}\|$ is the *length of* the vector $\mathbf{w}$:

$$\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}.$$

Just **Pythagoras** in n dimensions.

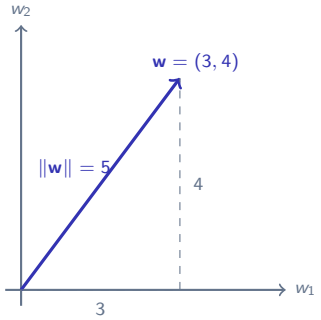
Worked example 1. $\mathbf{w} = (3, 4)$:

$$\|\mathbf{w}\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5.$$

Worked example 2. $\mathbf{w} = (1, 1)$:

$$\|\mathbf{w}\| = \sqrt{1^2 + 1^2} = \sqrt{2} \approx 1.414.$$

Note. Also written $\|\mathbf{w}\|_2$. Always ≥ 0, equals 0 only for $\mathbf{w} = \mathbf{0}$.



SVM link. Margin $= 1/\|\mathbf{w}\|$. **Smaller $\|\mathbf{w}\| \Rightarrow$ wider margin.** That's why F2 minimises $\|\mathbf{w}\|^2$.

$\|\mathbf{w}\|$ = length of w via Pythagoras. The SVM objective shrinks this to widen the margin.

Worked Example Step 1: Is $(0, 0)$ Classified Correctly?

Candidate line. $\mathbf{w} = (1, 1)$, $b = -1$. Boundary: $x_1 + x_2 = 1$.

Plug in $(0, 0)$**.**

$$\mathbf{w} \cdot (0, 0) + b = 1 \cdot 0 + 1 \cdot 0 + (-1) = -1.$$

Sign. Negative.

Prediction. Class -1.

True label. $y = -1$.

Match! The candidate line classifies $(0, 0)$ correctly.

Why this matters.

We are checking that the candidate line satisfies the SVM's constraint: every training point must be on the correct side.

Not only correct, but satisfying

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

(we will see this constraint in the primal formulation on a later slide).

At $(0, 0)$: $y_i \cdot (\mathbf{w} \cdot \mathbf{x} + b) = (-1) \cdot (-1) = +1 \geq 1$.
Tight.

Step 1 verified: $(0, 0)$ is on the negative side, and the margin constraint is satisfied with equality.

Worked Example Step 2: Is $(1, 1)$ Classified Correctly?

Same candidate line. $\mathbf{w} = (1, 1)$, $b = -1$.

Plug in $(1, 1)$**.**

$$\mathbf{w} \cdot (1, 1) + b = 1 + 1 - 1 = 1.$$

Sign. Positive.

Prediction. Class $+1$.

True label. $y = +1$.

Match! Classified correctly.

Margin constraint check.

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = (+1) \cdot (+1) = +1 \geq 1.$$

Tight again. The constraint holds with equality, which means $(1, 1)$ sits exactly on the margin boundary.

What about $(2, 2)$**?**

$$(+1) \cdot (1 \cdot 2 + 1 \cdot 2 - 1) = (+1) \cdot (3) = 3 \geq 1.$$

Not tight — $(2, 2)$ is strictly past the margin, so it will not be a support vector.

Step 2 verified: $(1, 1)$ **is on the positive side; it sits exactly on the upper margin boundary.**

Worked Example Step 3: Compute the Margin

Formula. Margin $= 1/\|\mathbf{w}\|$.

Plug in $\mathbf{w} = (1, 1)$.

$$\|\mathbf{w}\| = \sqrt{1^2 + 1^2} = \sqrt{2}.$$

$$\text{margin} = \frac{1}{\sqrt{2}} \approx 0.707.$$

Interpretation. Every training point sits *at least* $1/\sqrt{2} \approx 0.707$ units away from the boundary (measured perpendicular).

Points that sit EXACTLY at distance $1/\sqrt{2}$: $(0, 0)$ and $(1, 1)$. The point $(2, 2)$ sits farther.

Where does the formula margin $= 1/\|\mathbf{w}\|$ come from?

From the distance formula: for a point on the margin $(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1)$,

$$\text{dist} = \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}.$$

The same value on each side, so total margin (width between the two parallel margin lines) is $2/\|\mathbf{w}\|$.

Take-away. Smaller $\|\mathbf{w}\|$ means wider margin.

Margin $= 1/\|\mathbf{w}\| = 1/\sqrt{2} \approx 0.707$. Smaller $\|\mathbf{w}\|$ would give a wider margin.

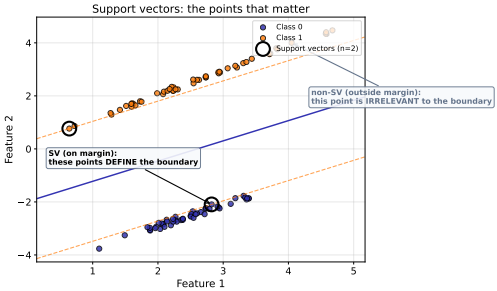
Support Vectors: The Points That Matter

Definition. A **support vector** is a training point that sits EXACTLY on one of the two margin boundaries.

In our worked example.

- $(0, 0)$: on the lower margin (dist $1/\sqrt{2}$, class -1)
- $(1, 1)$: on the upper margin (dist $1/\sqrt{2}$, class $+1$)
- $(2, 2)$: strictly past the upper margin, so *not* a support vector

Only two support vectors here. In general, on a d-dimensional problem, you need at least $d + 1$ support vectors.



Support vectors = training points touching the margin. Non-SVs can be removed without changing anything.

Why Support Vectors Matter

The sparse property. Remove any training point that is *not* a support vector. Retrain. You will get the **same** SVM.

In our worked example:

- Remove $(2, 2)$. Re-solve with just $(0, 0)$ and $(1, 1)$.
- You will again get $\mathbf{w} = (1, 1)$, $b = -1$.
- Only the support vectors $(0, 0)$ and $(1, 1)$ constrain the solution.

Consequence. The decision boundary depends *only* on the support vectors, not on the bulk of the training data.

Why this is great.

- The trained SVM is **sparse**: storage is proportional to the number of support vectors, not the full training set
- Prediction is fast: only dot products with support vectors are needed
- Robust to irrelevant points far from the boundary — they have zero influence

Consequence for noisy data. Points well inside the correct class territory do not disturb the SVM solution.

Remove any non-SV: the SVM is unchanged. This sparsity is a defining feature of SVMs.

Why Do We Need the Max-Margin Idea?

The nightmare scenario. Imagine the boundary is razor-thin — every training point barely misses it.

What can go wrong?

- A new data point arrives, shifted slightly by noise
- It crosses the nearly-touching boundary and is misclassified
- Small input perturbations $\Rightarrow$ big prediction flips

Wide-margin cure. If the boundary is d units away from every training point, then any perturbation smaller than d cannot cause a misclassification on those points.

Wide margin $\Rightarrow$ robust classifier $\Rightarrow$ good generalisation
(**generalisation** = performance on *unseen data* — data not seen during training).

Theoretical backing.

This intuition is made rigorous in statistical learning theory. Vapnik proved generalisation bounds of the form

$$\text{test error} \leq \text{train error} + O\left(\frac{1}{\text{margin}}\right)$$

(schematic).

Bigger margin $\Rightarrow$ tighter generalisation bound. The guarantee is the whole reason the method was developed.

It is also why SVM was the dominant tabular classifier for two decades (1995–2015).

Wide margin = robust to noise. This is also backed by formal generalisation bounds (bounds on test error).

Bias & Variance in 1 Slide

Two ways a model can fail.

Bias = *systematic error*; how far the average prediction sits from the truth.

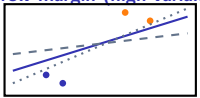
- Lines that wobble every which way are high-bias only if they are systematically wrong
- A too-simple model (straight line for curved data) has high bias

Variance = how much predictions *jitter* across different training sets.

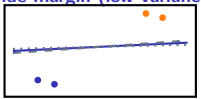
- A model that changes wildly when you swap a few rows has high variance
- Narrow-margin lines often have high variance: a single noisy point flips the boundary

SVM's wide-margin rule reduces variance: small changes in data barely move a wide-margin boundary.

Narrow margin (high variance)



Wide margin (low variance)



In short. Wide margin ⇒ *low variance*, not too much bias ⇒ good test error.

Bias = systematic error. Variance = jitter across training sets. SVM trades a bit of bias for a lot of variance reduction.

Hard-Margin SVM: The Full Statement

Goal in plain English.

Find the line with the *maximum* margin **subject to** classifying every training point correctly.

Two demands at once.

1. All training points on the correct side (*zero training error*)
2. Among all such lines, pick the one with the widest margin

What if no such line exists? Then the problem is *infeasible* and hard-margin SVM has no solution. We will fix this with *soft margin* in a few slides.

Two equivalent ways to say "widest margin".

- **Maximise** $2/\|\mathbf{w}\|$
- **Minimise** $\|\mathbf{w}\|$
- **Minimise** $\frac{1}{2}\|\mathbf{w}\|^2$ (same minimisers, nicer derivatives)

The last form is what the textbook primal uses. The $\frac{1}{2}$ is a convention that cancels out when we take derivatives.

Wide margin + zero training error. These two demands pin down a unique solution when the data is separable.

Hard-Margin Primal (Formula F2)

Formula F2.

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

$$\text{s.t.} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \ \ \forall i.$$

Read it piece by piece.

- Minimise $\|\mathbf{w}\|^2$ (maximises margin)
- Over all choices of $\mathbf{w}$ and b
- Subject to: every training point sits on the correct side *at distance* $\geq 1/\|\mathbf{w}\|$

This is a **quadratic programme** (**QP** = quadratic objective with linear constraints, solved by standard algorithms). **Convex** = shaped like a bowl, one global minimum (no other valleys), so any local optimiser finds it. Always has a unique solution if data is separable.

The constraint decoded.

$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ demands two things at once:

- Correct side: $\mathbf{w} \cdot \mathbf{x}_i + b$ has the same sign as y_i
- Sufficiently far: the product is ≥ 1, not just > 0

The "≥ 1" rather than "> 0" fixes the scale of $\mathbf{w}$. Without it, you could shrink $\mathbf{w}$ arbitrarily and get any margin you like.

F2 = the hard-margin primal. Quadratic objective, linear constraints. Solved by standard QP solvers.

Worked Example: F2 on Our 3-Point Setup

Plug our candidate into F2.

Data: $(1,1), (2,2)$ $[+1]$; $(0,0)$ $[-1]$.

Candidate: $\mathbf{w} = (1,1), b = -1$.

Objective value.

$$\tfrac{1}{2}\|\mathbf{w}\|^2 = \tfrac{1}{2}(1^2 + 1^2) = \tfrac{1}{2} \cdot 2 = \mathbf{1}.$$

Check all 3 constraints $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$.

- $(1,1)$: $(+1)(1+1-1) = 1 \geq 1$ ✓(tight)
- $(2,2)$: $(+1)(2+2-1) = 3 \geq 1$ ✓
- $(0,0)$: $(-1)(0+0-1) = 1 \geq 1$ ✓(tight)

Feasible. Objective $= 1$.

Why is this the minimum?

Try shrinking $\mathbf{w}$, say $\mathbf{w} = (0.9, 0.9)$. The tight constraints fail:

$$(+1)(0.9 + 0.9 - 1) = 0.8 < 1 \ \text{X}.$$

Try growing $\mathbf{w}$, say $\mathbf{w} = (2,2)$. Feasible but

$$\tfrac{1}{2}(2^2 + 2^2) = 4 > 1.$$

Bigger objective.

Conclusion. $\mathbf{w} = (1,1), b = -1$ is the F2 minimum. Objective 1. Margin $= 1/\|\mathbf{w}\| = 1/\sqrt{2}$.

What this shows. F2 is not abstract — you can verify its solution on a tiny dataset by hand.

$\frac{1}{2}\|\mathbf{w}\|^2 = 1$ **at the optimum. All constraints satisfied, two of them tight. Margin** $= 1/\sqrt{2}$.

What F2 Says in Plain English

Strip all the symbols away.

F2 says:

"Minimise $\|\mathbf{w}\|$ (which maximises margin because margin $= 1/\|\mathbf{w}\|$), subject to keeping every point at least one margin-unit away on the correct side."

Two parts:

- Part A (objective): make $\|\mathbf{w}\|$ as small as we can
- Part B (constraint): but do not let any training point cross into the "wrong" zone

Part A wants to shrink $\|\mathbf{w}\|$. Part B says "not too much". The answer sits at the sweet spot.

Geometric picture.

Imagine the two parallel margin lines are rails. You push the rails apart (smaller $\|\mathbf{w}\|$) until they are about to collide with a training point.

The point that stops the widening is a **support vector**.

In our worked example: the rails are $x_1 + x_2 = 0$ (through the -1 class) and $x_1 + x_2 = 2$ (through the $+1$ class). They are stopped by $(0, 0)$ and $(1, 1)$ respectively.

F2 in one sentence: push the margin rails apart until a data point stops them.

Why Minimise $\|\mathbf{w}\|^2$ (not $\|\mathbf{w}\|$)?

The short answer.

Margin $= 1/\|\mathbf{w}\|$, so *minimising* $\|\mathbf{w}\|$ *maximises margin*. Both $\|\mathbf{w}\|$ and $\|\mathbf{w}\|^2$ have the same minimiser because squaring is monotonic for non-negative values.

Why then switch to $\|\mathbf{w}\|^2$?

- $\|\mathbf{w}\|^2 = \mathbf{w} \cdot \mathbf{w}$ is smooth (differentiable everywhere)
- $\|\mathbf{w}\| = \sqrt{\mathbf{w} \cdot \mathbf{w}}$ has a kink at $\mathbf{w} = 0$
- Smooth objectives are easier to optimise (closed-form gradients, nicer solvers)

Why the factor of $\frac{1}{2}$?

Pure bookkeeping. The derivative of $\frac{1}{2}\|\mathbf{w}\|^2$ with respect to $\mathbf{w}$ is just $\mathbf{w}$ (no stray factor of 2).

Without the $\frac{1}{2}$, the dual formulation would have a 2 floating around in front of the quadratic term. The convention eliminates it.

Equivalent problem. Whether you minimise $\|\mathbf{w}\|$, $\|\mathbf{w}\|^2$, or $\frac{1}{2}\|\mathbf{w}\|^2$, you get the *same* optimal $\mathbf{w}$.

$\|\mathbf{w}\|^2$ is smoother than $\|\mathbf{w}\|$. The $\frac{1}{2}$ is bookkeeping that cancels in the derivative.

Real Data Is Messy

The hard-margin assumption fails. In practice:

- Classes overlap due to measurement noise
- A handful of outliers sit on the wrong side
- No straight line achieves zero training error

What happens to hard-margin SVM? The constraints $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ become inconsistent. The optimisation has **no feasible solution**.

The fix. Allow some constraints to be violated, but charge a penalty. This is **soft-margin SVM**.

Real-world examples that need soft margin.

- Credit scoring: a few high-income borrowers still default
- Fraud detection: some legitimate transactions look like fraud
- Medical diagnosis: symptoms overlap between diseases

The core idea. We will introduce a "slack" variable ξ_i for each training point. It measures *how badly* point i violates its margin constraint.

No violation $\Rightarrow \xi_i = 0$.

Overlap, outliers, noise. Hard-margin fails. We need slack variables to make the problem feasible.

Slack Variables ξ_i Explained

Definition. ξ_i = how far point i falls inside the margin (or on the wrong side), measured in the same units as the margin.

Three regimes for a point i.

- **Past the margin** on the correct side: $\xi_i = 0$. No violation.
- **Inside the margin** on the correct side: $0 < \xi_i \leq 1$. Partial violation.
- **Wrong side** of the boundary: $\xi_i > 1$. Misclassified.

Non-negativity. $\xi_i \geq 0$ always. There is no "bonus" for being far past the margin.

Formal definition.

$\xi_i = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$.

That is exactly the *hinge loss* (next slide).

Concrete example. Suppose $y_i = +1$ and $\mathbf{w} \cdot \mathbf{x}_i + b = 0.3$.

$$\xi_i = \max(0, 1 - 0.3) = 0.7.$$

The point is on the correct side but inside the margin by 0.7.

ξ_i measures the violation. Zero if the point is safely past the margin. Larger if misclassified.

Soft-Margin Primal (Formula F3)

Formula F3.

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

$$\text{s.t.} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \ \ \xi_i \geq 0.$$

Two objectives, one sum.

- $\frac{1}{2}\|\mathbf{w}\|^2$: keep margin wide
- $C\sum_i \xi_i$: penalise slack (violations)

C is the **knob** that balances the two. Larger C: bigger penalty per violation.

What changed from F2 to F3?

- The constraint relaxed from "≥ 1" to "$\geq 1 - \xi_i$", allowing violation amount ξ_i
- A new non-negativity constraint: $\xi_i \geq 0$
- The objective picks up $C\sum_i \xi_i$ to discourage large slacks

Recovers hard-margin. As $C \to \infty$, no slack is tolerated, F3 reduces to F2.

Strength. F3 is convex too; standard QP solvers handle it.

F3 = F2 with slack variables ξ_i and penalty $C\sum_i \xi_i$. $C \to \infty$ recovers hard-margin.

Worked Example: One Violating Point and C

Same setup, with a troublemaker.

Original: $(1,1), (2,2)$ $[+1]$; $(0,0)$ $[-1]$. **Add:** $(0.5, 0.5)$ $[+1]$.

At $\mathbf{w} = (1,1), b = -1$:

$$(+1)(0.5 + 0.5 - 1) = 0 \text{ violates } \geq 1.$$

So we need slack $\xi = 1 - 0 = 1$.

F3 objective.

$$\tfrac{1}{2}\|\mathbf{w}\|^2 + C\,\xi = 1 + C \cdot 1.$$

Sweep C.

- $C = 0.01$: objective $= 1.01$, slack cheap, keep wide margin
- $C = 1$: objective $= 2$, balanced
- $C = 1000$: objective $= 1001$, huge penalty $\Rightarrow$ optimiser will reshape $\mathbf{w}$ to remove ξ

What the optimiser does as C grows.

- Small C (0.01): tolerate $\xi = 1$ and *keep* wide margin. Boundary stays at $x_1 + x_2 = 1$.
- Large C (1000): optimiser tilts the boundary to include the troublemaker $(0.5, 0.5)$. Margin shrinks.

The key insight. C converts a geometric question ("how wide should the margin be?") into a trade-off between margin width and total slack.

Total slack $\sum_i \xi_i$ = sum of individual violations. Even a single $\xi_i = 1$ (i.e., the point sits on the decision boundary) costs C.

One violator, two C extremes. $C = 0.01$: wide margin, big slack. $C = 1000$: narrow margin, zero slack.

What Does C Mean?

C = **how much we PENALISE each slack.**

Two extremes.

- **Small** C (e.g. 0.01): the penalty for slack is cheap. The optimiser happily tolerates many violations if that lets $\|\mathbf{w}\|$ shrink. Result: *wide margin, more misclassified training points*.
- **Large** C (e.g. 1000): violations are expensive. The optimiser fights to make $\xi_i = 0$ for every point. Result: *narrow margin, few misclassified training points*.

In practice. C is a **hyperparameter** (a setting of the model chosen *BEFORE training*, not learned from data) tuned by **cross-validation** (split training data into k folds; train on $k-1$ folds, measure on the held-out fold; average) on a log scale (**log scale** = logarithmic spacing, multiply by 10 between grid points, powers of 10).

Rule of thumb.

- **Noisy data**, many overlapping points: small C
- **Clean data**, few outliers: large C
- **Don't know**: start with $C = 1$, sweep log grid $\{10^{-2}, 10^{-1}, 1, 10, 10^{2}\}$

Bias-variance. (Bias = systematic error; variance = jitter across training sets.)

- Large C: low bias (fits training), high variance (jitter, overfit risk)
- Small C: high bias (systematic error, underfits), low variance (robust)

C sets the tradeoff: wide margin + many violations vs narrow margin + few violations.

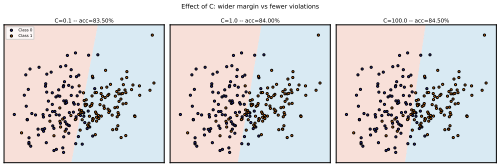
Effect of C Visualised

Three panels, three values of C.

- $C = 0.1$: wide margin, many slack-admitted violations, smoother boundary
- $C = 1$: balanced between width and fit
- $C = 100$: narrow margin, boundary fights to classify every training point

Notice. Moving from left to right, the margin shrinks visibly. The boundary also gets "noisier" as it tries to accommodate individual points.

Which is best? The middle panel usually generalises best; the extremes tend to under-fit or over-fit.



As C grows, the margin shrinks and the boundary chases individual points. Classic bias-variance knob.

Hinge Loss (Formula F4)

Formula F4.

$$L(y, f(\mathbf{x})) = \max(0,\ 1 - y \cdot f(\mathbf{x})).$$

Here $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ (the *raw* score, not the sign).

Three cases.

- $y \cdot f(\mathbf{x}) \geq 1$: past the margin on the correct side. Loss $= 0$.
- $0 \leq y \cdot f(\mathbf{x}) < 1$: correct side but inside margin. Loss $= 1 - y \cdot f(\mathbf{x}) \in (0, 1]$.
- $y \cdot f(\mathbf{x}) < 0$: wrong side. Loss > 1, grows linearly.

The hinge.

The function has a corner ("hinge") at $y \cdot f = 1$. Before the hinge, loss is positive and linear. After the hinge, loss is exactly zero.

Equivalent form of F3. Soft-margin SVM is the same as

$$\min_{\mathbf{w},b} \tfrac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \max(0, 1 - y_i f(\mathbf{x}_i)).$$

This is called the *unconstrained hinge-loss view* of SVM.

Hinge loss = max-margin penalty. Zero past the margin, linear otherwise. The defining loss of SVM.

Numeric Check: Hinge Loss Table

Fix $y = +1$. Plug in seven raw scores $f(\mathbf{x})$ and compute $L = \max(0, 1 - y \cdot f)$.

$f(\mathbf{x})$	$1 - y \cdot f$	L
-1.0	2.0	**2.0** (misclassified)
-0.5	1.5	**1.5** (misclassified)
0.0	1.0	**1.0** (on boundary)
0.5	0.5	**0.5** (inside margin)
1.0	0.0	**0.0** (on margin)
1.5	-0.5	**0.0** (past margin)
2.0	-1.0	**0.0** (past margin)

Key observation. Once $f \geq 1$ the loss is *flat zero past the margin*. Before, it rises linearly with slope $-y = -1$.

What this shows.

- Confident *correct* predictions (large f, same sign as y) are free
- Confident *wrong* predictions are charged heavily
- No gradient for easy examples past the margin $\Rightarrow$ SVM ignores them

Compare with log loss at the same points:
$\log(1 + e^{-yf})$ gives
$\{1.31, 0.97, 0.69, 0.47, 0.31, 0.20, 0.13\}$ — *never zero*.

This asymmetry is how SVM sparsity emerges. Only points with $L > 0$ (support vectors or violators) affect the gradient and thus $\mathbf{w}$.

Seven scores, seven hinge losses. Once $y \cdot f \geq 1$, loss drops to exactly zero.

What Hinge Loss Means in Practice

Key distinguishing feature.

If $y \cdot f(\mathbf{x}) \geq 1$, loss is zero.

Compare with *log loss* (logistic regression):

$$L_{\log} = \log(1 + e^{-yf(\mathbf{x})})$$

which is *always positive*, even for points far past the margin.

Consequence for the SVM.

- Points well inside the correct region contribute nothing to the loss or the gradient
- Only points on or inside the margin (i.e., support vectors) matter
- This is *how* sparsity emerges from the loss function

Why the asymmetry matters.

Imagine a confident, correct prediction far past the margin. Log loss still rewards pushing it *even* further. Hinge loss does not.

This means SVM spends its optimisation effort on the *hard* cases, not on polishing the easy ones.

Analogy. A teacher who spends all her time on the struggling students, not on the A+ students. Hinge loss is a teacher like that.

Points past the margin have zero loss. SVM only cares about examples near or on the wrong side.

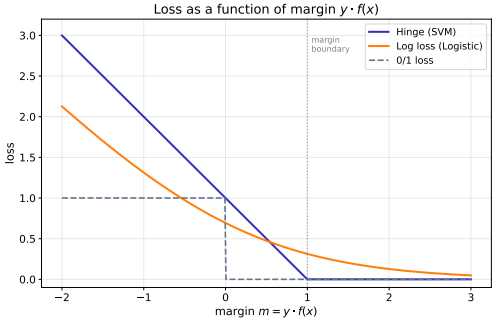
Loss Comparison: Hinge vs Log vs 0/1

Three loss functions side by side.

- **Hinge (SVM)**: flat zero past the margin, then linear
- **Log loss (logistic)**: smooth, never exactly zero
- 0/1 **loss**: step function, non-convex, can't be optimised directly

Both hinge and log are convex surrogates for 0/1 (a **surrogate** replaces 0/1 loss with a function that is easier to optimise). They upper-bound 0/1 and are tractable.

Key difference. Hinge reaches zero at the margin; log is always positive. This is the root cause of SVM's sparsity.



Hinge drops to zero past the margin. Log loss never does. The shape of the loss explains SVM's sparsity.

What Is the Dual?

Same problem, different variables.

The **primal** (F3) is the *problem as written*, the original problem in terms of $\mathbf{w}, b, \xi$. The **dual** is an *equivalent rewrite* in terms of one **Lagrange multiplier** $\alpha_i \geq 0$ per training point.

The derivation uses Lagrangian duality and KKT conditions; we will not reproduce it here, just state the result and its consequences.

The key change.

In the dual, the data appears only through dot products $\mathbf{x}_i \cdot \mathbf{x}_j$.

That observation is the seed of the kernel trick.

Why rewrite at all?

- The dual makes sparsity explicit: most $\alpha_i = 0$
- The dual number of variables equals the number of training points (not the dim of $\mathbf{x}$)
- The dual depends on inputs only through dot products — lets us generalise to kernels

Optimisation. Dual is still convex QP, solvable by **SMO** (**Sequential Minimal Optimisation** — the specialised algorithm that solves the SVM dual efficiently by optimising two α at a time).

Primal in $\mathbf{w}, b, \xi$ $\Leftrightarrow$ dual in α_i. Same problem, but the dual uses dot products only.

What Is a Lagrangian?

The big idea. The **Lagrangian** is a *rewrite of a constrained optimisation as an unconstrained one*, adding one new variable α per constraint. It combines the objective with constraints, paying prices on constraints for violations.

General form (for "minimise $f(x)$ s.t. $g(x) \leq 0$"):

$$L(x, \alpha) = f(x) + \alpha \underbrace{g(x)}_{\leq 0 \text{ constraint}}, \quad \alpha \geq 0.$$

Read it.

- $\alpha = 0$: constraint inactive; Lagrangian $= f(x)$ alone
- $\alpha > 0$: constraint is binding; α is the *price* for violating

This is a **penalty function** that automatically tightens up when constraints threaten to be violated.

Tiny example.

Minimise x^2 subject to $x \geq 1$ (i.e., $1 - x \leq 0$).

$$L(x, \alpha) = x^2 + \alpha(1 - x).$$

Set $\partial L / \partial x = 0$: $2x - \alpha = 0 \Rightarrow x = \alpha/2$.

Plug into the constraint at equality ($1 - x = 0$): $x = 1$, hence $\alpha = 2$.

Result. The solution is $x = 1, \alpha = 2$. Objective value $= 1$. The price $\alpha = 2$ is exactly the derivative of the optimal value with respect to the constraint bound.

For SVM. Each training point gets one $\alpha_i \geq 0$ — its price for the margin constraint.

Lagrangian = objective + prices on constraints. One extra variable α per constraint. Solves by setting derivatives to zero.

SVM Lagrangian (Full Form)

The full soft-margin Lagrangian (introducing one α_i per margin constraint, one μ_i per $\xi_i \geq 0$):

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \left[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i\right] - \sum_i \mu_i \xi_i.$$

Read each term.

- $\frac{1}{2}\|\mathbf{w}\|^2$: *objective* (widen margin)
- $C\sum_i \xi_i$: *slack cost* (penalise violations)
- $-\sum_i \alpha_i[\cdots]$: *constraint prices* for margin constraints, $\alpha_i \geq 0$
- $-\sum_i \mu_i \xi_i$: *non-negativity prices* on slack, $\mu_i \geq 0$

Why the minus signs?

The bracket $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i \geq 0$ must be non-negative. When feasible, the bracket is ≥ 0 and the product $-\alpha_i[\text{bracket}] \leq 0$. Lagrangian encourages $\alpha_i > 0$ only when a constraint is tight.

Next steps (two slides from here).

1. Set $\partial L/\partial(\mathbf{w}, b, \xi) = 0$ to get first-order conditions
2. Substitute back into L to eliminate $(\mathbf{w}, b, \xi)$
3. What's left is the dual (F5)

Preview. The dual depends on the data only through dot products $\mathbf{x}_i \cdot \mathbf{x}_j$.

Four pieces: objective, slack cost, constraint prices, non-negativity prices. One α_i per point.

First-Order Conditions

Set derivatives of L to zero.

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

Caption. $\mathbf{w}$ is forced to be a **linear combination of support vectors**. Gives the weight recovery $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$.

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0$$

Caption. The balance $\sum_i \alpha_i y_i = 0$ keeps total positive mass equal to negative mass.

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0$$

Caption. α stays between 0 and C (written formally $0 \le \alpha_i \le C$): $\alpha_i = C - \mu_i$ and both are non-negative.

What just happened.

Three stationarity equations give:

- $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$ (weight recovery)
- $\sum_i \alpha_i y_i = 0$ (class-balance)
- $0 \le \alpha_i \le C$ (box constraint)

Each coefficient α_i is non-negative because it is a constraint price. And each is capped at C because of the slack-price coupling $\alpha_i = C - \mu_i$.

These three equations are the KKT stationarity conditions. The next slide substitutes them back into L to eliminate $(\mathbf{w}, b, \xi)$.

Three stationarity equations: w-recovery, class balance, box $0 \le \alpha_i \le C$.

Plug First-Order Conditions Back to Get Dual

Substitute $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$ into L.

Step 1. Substitute w into L to eliminate **w**:

$$\tfrac{1}{2}\|\mathbf{w}\|^2 = \tfrac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j).$$

Caption. The quadratic in **w** becomes a quadratic in α, with data entering only as dot products.

Step 2. Cancel cross-terms using $C - \alpha_i - \mu_i = 0$ and $\sum_i \alpha_i y_i = 0$ — the b and ξ pieces drop out.

Step 3. Arrive at F5:

$$\max_{\boldsymbol{\alpha}} \sum_i \alpha_i - \tfrac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

with $0 \le \alpha_i \le C$ and $\sum_i \alpha_i y_i = 0$.

What we bought.

- Primal had $d + 1 + n$ variables $(\mathbf{w}, b, \xi)$
- Dual has n variables (α_i)
- Data appears *only* as dot products

This is the seed of the kernel trick. Anywhere $\mathbf{x}_i \cdot \mathbf{x}_j$ appears, we can replace it with $K(\mathbf{x}_i, \mathbf{x}_j)$ and get nonlinear SVM for free.

What F5 is. A quadratic programme in α, n variables, box + equality constraints. Still convex. Solvable by SMO.

Substitute $\Rightarrow$ cancel cross-terms $\Rightarrow$ Arrive at F5. Variables drop from $(\mathbf{w}, b, \xi)$ to just α.

Dual Formulation (Formula F5)

Formula F5.

$$\max_{\boldsymbol{\alpha}} \sum_i \alpha_i - \tfrac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{s.t. } 0 \le \alpha_i \le C, \quad \sum_i \alpha_i y_i = 0.$$

Read the objective.

- First term: reward raising the α_i
- Second term: penalty for pairs of same-class points with aligned $\mathbf{x}_i$ and $\mathbf{x}_j$

Read the constraints.

- $0 \le \alpha_i \le C$: bounded (upper bound ties to soft margin)
- $\sum_i \alpha_i y_i = 0$: class-balance constraint (emerges from $\partial/\partial b$)

Recovering w from α.

Once the dual is solved, the primal weights follow:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i.$$

Points with $\alpha_i = 0$ contribute nothing. Only the support vectors (those with $\alpha_i > 0$) contribute.

This is where sparsity becomes visible: the weight vector is a weighted sum of *only* the support vectors.

F5 = dual. Objective is quadratic in α; data appears only as $\mathbf{x}_i \cdot \mathbf{x}_j$. Remember this.

Worked Example: F5 on Our 3-Point Setup

Recall. $(1,1),(2,2)\,[+1]$; $(0,0)\,[-1]$. Hard-margin ($C \to \infty$).

From the KKT we expect only the margin-tight points ($(1,1)$ and $(0,0)$) to have $\alpha_i > 0$. Guess $\alpha_1 = \alpha_3 = 1$, $\alpha_2 = 0$ (with indices 1=$(1,1)$, 2=$(2,2)$, 3=$(0,0)$).

Check class balance $\sum_i \alpha_i y_i = 0$:

$$1 \cdot (+1) + 0 \cdot (+1) + 1 \cdot (-1) = 0. \quad \checkmark$$

Compute w via recovery:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i = 1 \cdot (1,1) + 1 \cdot (-1)(0,0) = (1,1). \quad \checkmark$$

Verify F5 objective value.

Dot products matrix (only indices with $\alpha > 0$):

$$\begin{pmatrix} \mathbf{x}_1 \cdot \mathbf{x}_1 & \mathbf{x}_1 \cdot \mathbf{x}_3 \\ \mathbf{x}_3 \cdot \mathbf{x}_1 & \mathbf{x}_3 \cdot \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}.$$

F5 objective:

$$\begin{aligned} &\sum_i \alpha_i - \tfrac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ &= (1+1) - \tfrac{1}{2} \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 2 \\ &= 2 - 1 = \mathbf{1}. \end{aligned}$$

This matches F2's primal objective $\frac{1}{2}\|\mathbf{w}\|^2 = 1$.
Strong duality confirmed.

$\alpha_1 = \alpha_3 = 1, \alpha_2 = 0$. **Balance OK. w recovered as** $(1,1)$. **Dual objective = primal objective = 1.**

KKT Conditions and Support Vectors (Formula F6)

Formula F6 (consequence of KKT).

$$\alpha_i > 0 \iff \mathbf{x}_i \text{ is a support vector.}$$

Three regimes (soft-margin).

- $\alpha_i = 0$: point past the margin, non-SV
- $0 < \alpha_i < C$: point on the margin (a "free" SV)
- $\alpha_i = C$: point inside the margin or misclassified (a "bounded" SV)

Plain English. Most α_i are zero. Only support vectors have $\alpha_i > 0$, and only those points contribute to $\mathbf{w}$.

The four KKT conditions (necessary at any constrained optimum):

1. **Primal feasibility**: $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$.
2. **Dual feasibility**: $0 \leq \alpha_i \leq C$.
3. **Stationarity**: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$, $\sum_i \alpha_i y_i = 0$.
4. **Complementary slackness**: $\alpha_i \cdot [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] = 0$.

The 4th line is the *F6 iff*: if a constraint is *slack* (strictly > 1), its α_i must be 0; conversely $\alpha_i > 0$ forces the constraint to be tight.

F6 formalises the sparsity property we mentioned earlier: the solution depends only on the support vectors.

$\alpha_i > 0$ **iff the point is a support vector. KKT 4-box pins this down exactly.**

Numeric Check: Complementary Slackness

Check the KKT complementary slackness
$\alpha_i \cdot [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0$ on our 3-point setup.
$\mathbf{w} = (1, 1), b = -1$; hard-margin, so $\xi_i = 0$.
Point 1 $(1, 1)$, $y = +1$, $\alpha_1 = 1$.

$$y(\mathbf{w} \cdot \mathbf{x} + b) - 1 = 1 \cdot (1 + 1 - 1) - 1 = 0.$$

$\alpha_1 \cdot 0 = 0$. ✓
Point 2 $(2, 2)$, $y = +1$, $\alpha_2 = 0$.

$$y(\mathbf{w} \cdot \mathbf{x} + b) - 1 = 1 \cdot (2 + 2 - 1) - 1 = 2.$$

$0 \cdot 2 = 0$. ✓
Point 3 $(0, 0)$, $y = -1$, $\alpha_3 = 1$.

$$y(\mathbf{w} \cdot \mathbf{x} + b) - 1 = (-1) \cdot (0 + 0 - 1) - 1 = 0.$$

$\alpha_3 \cdot 0 = 0$. ✓

Reading the pattern.

- **Tight constraint** $\Rightarrow \alpha_i$ *may* be positive (points 1, 3): they are support vectors
- **Slack constraint** $\Rightarrow \alpha_i$ *must* be zero (point 2): non-SV

This is F6 in action. The slackness condition forces α_i to zero for points past the margin, concentrating all non-zero α's on the support vectors.

Consequence. For the prediction $f(\mathbf{x}) = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$, we need only points 1 and 3.

All three KKT products equal 0. SV-ness is exactly the $\alpha_i > 0$ condition, nothing more.

The Kernel Trick: The Big Idea

The observation. In the dual (F5) and in the resulting decision function, the training data $\mathbf{x}_i, \mathbf{x}_j$ appear *only* as dot products $\mathbf{x}_i \cdot \mathbf{x}_j$.

The trick. Replace $\mathbf{x} \cdot \mathbf{x}'$ with a **similarity function** $K(\mathbf{x}, \mathbf{x}')$ (**similarity function** = a function that measures how similar two inputs are; larger K = more similar). We never need $\mathbf{x}$ itself anywhere; we swap dot product for K and never compute φ explicitly.

Why this is astonishing.

- We get nonlinear classification without changing the algorithm
- No explicit feature map φ needs to be computed
- The only condition on K: it must be a valid kernel (positive semi-definite)

What does "valid kernel" mean?

There exists some function φ such that $K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}')$. The feature space is implicit.

Example. The RBF kernel corresponds to an *infinite-dimensional* φ. We would never compute it, but we use it via K.

Mercer's theorem characterises which functions are valid kernels. For this course: just use the three classic ones.

Kernel trick: swap dot products for $K(\mathbf{x}_i, \mathbf{x}_j)$. Nonlinear classification, no extra computational cost.

Feature Map φ Explained

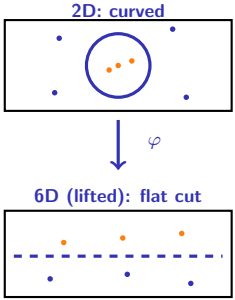
Feature map φ = a function that *lifts the input* into a *higher-dimensional space*, where curved boundaries become flat.

Explicit example (for degree-2 polynomial kernel):

$$\varphi(\mathbf{x}) = (1,\ x_1,\ x_2,\ x_1^2,\ x_2^2,\ x_1 x_2).$$

A 2D point lifts to 6D. A quadratic boundary in 2D (e.g., the circle $x_1^2 + x_2^2 = 1$) becomes linear in 6D: just zero out the 1, x_1^2, x_2^2 coefficients appropriately.

Kernel trick, restated. We *never compute* φ; we just use $K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}')$. For the RBF kernel, φ is infinite-dimensional — uncomputable but usable via K.



Curved in 2D ⇒ linear in the higher-dim feature space.

φ lifts the input to a higher-dimensional space where data is linearly separable. Kernel trick lets us skip φ.

PSD & Mercer in Plain English

PSD (positive semi-definite) = a property of a symmetric matrix K ensuring

$$z^T K z = \underbrace{\sum_{ij} K_{ij} z_i z_j}_{\text{sum is never negative}} \geq 0$$

for every vector z. Equivalently: all eigenvalues of K are ≥ 0.

Mercer's theorem (plain English). $K(\mathbf{x}, \mathbf{x}')$ is a *valid kernel iff* the Gram matrix $[K(\mathbf{x}_i, \mathbf{x}_j)]$ is PSD for every choice of points, iff K is an inner product in some feature space φ.

What this bought. Given any PSD K, there exists a (possibly infinite-dim) φ such that $K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}')$. The kernel trick is justified.

Numeric check: RBF 2×2 is PSD.

Take $\mathbf{x}_1 = (0, 0), \mathbf{x}_2 = (1, 0)$, $\gamma = 1$.

$$K = \begin{pmatrix} 1 & e^{-1} \\ e^{-1} & 1 \end{pmatrix}.$$

Eigenvalues. Trace $= 2$, determinant $= 1 - e^{-2}$. So the eigenvalues $1 \pm e^{-1}$:

$$\lambda = 1 \pm e^{-1} \approx \{1.368, 0.632\}.$$

Both **eigenvalues $1 \pm e^{-1}$ are positive** $\Rightarrow$ K is PSD. ✓

Take-away. "Sum is never negative" is the working test: Mercer says this check is exactly what makes a function a legitimate kernel.

PSD: $z^T K z \geq 0$. Mercer: valid kernel iff PSD iff inner product. Eigenvalue check is direct.

Kernelised Decision Function (Formula F7)

Formula F7.

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right).$$

Read it piece by piece.

- Sum runs over *all training points* (but only SVs have $\alpha_i > 0$)
- Each SV contributes $\alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$
- Take the sign of the total plus b to get the label

Plain English. "Weighted vote of the support vectors, weighted by how similar (via K) the query is to each."

Linking F1 to F7.

If $K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$ (the linear kernel), then

$$\sum_i \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) = \left(\sum_i \alpha_i y_i \mathbf{x}_i\right) \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{x}.$$

So F7 with a linear kernel reduces to F1: $\operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$.

For nonlinear kernels, the $\mathbf{w}$ vector lives in an implicit feature space and cannot be written down directly — but F7 still works.

F7 = sign of similarity-weighted vote over support vectors. Reduces to F1 for the linear kernel.

Worked Example: F7 on Our 3-Point Setup

Setup. From our 3-point worked example: $\alpha_1 = \alpha_3 = 1$, $\alpha_2 = 0$. Points $\mathbf{x}_1 = (1,1)$ $[+1]$, $\mathbf{x}_3 = (0,0)$ $[-1]$, $b = -1$. Linear kernel $K(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$.

Classify a test point $\mathbf{x} = (3,3)$.

Term 1 ($i = 1$):

$$\alpha_1 y_1 K(\mathbf{x}_1, \mathbf{x}) = 1 \cdot 1 \cdot (1 \cdot 3 + 1 \cdot 3) = 6.$$

Term 2 ($i = 2$): $\alpha_2 = 0$, so contribution $= 0$.

Term 3 ($i = 3$):

$$\alpha_3 y_3 K(\mathbf{x}_3, \mathbf{x}) = 1 \cdot (-1) \cdot (0 \cdot 3 + 0 \cdot 3) = 0.$$

Sum + bias.

$$\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b = 6 + 0 + 0 - 1 = \mathbf{5}.$$

Sign. $+5 > 0 \Rightarrow$ predict class $+\mathbf{1}$.

Sanity check vs F1. With $\mathbf{w} = (1,1), b = -1$:

$$\mathbf{w} \cdot (3,3) + b = 3 + 3 - 1 = 5. \ \checkmark$$

Same number, same prediction.

What this shows. F7 is just a rewrite of F1 for the linear kernel. The extra machinery (the $\alpha_i y_i$ sum) lets us swap in any other kernel later without changing any other line of code.

$\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b = 5 > 0 \Rightarrow +1$. **F7 reproduces F1's answer for the linear kernel.**

The Three Classic Kernels

Three kernels you will see in practice.

- **Linear**: plain dot product, no transformation
- **Polynomial**: maps to polynomial features up to degree d
- **RBF (radial basis function)**: similarity decays with distance, infinite-dim feature space

When to reach for each.

- Linear: many features, mostly linear boundary, big data
- Polynomial: you suspect specific interactions (pairs, triples)
- RBF: general-purpose nonlinear, medium-size data — the default

Formulas preview.

- **Linear**: $K = \mathbf{x} \cdot \mathbf{x}'$
- **Polynomial**: $K = (\mathbf{x} \cdot \mathbf{x}' + c)^d$
- **RBF**: $K = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$

Each one has hyperparameters (d, c, γ) that control the geometry of the implicit feature space. Tuning is via cross-validation.

We will plug each into a single worked example in the next three slides.

Three kernels, three feature spaces, three hyperparameters. Linear, polynomial, RBF.

Linear Kernel (Formula F8)

Formula F8.

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'.$$

Plain English. Original dot product. No transformation. Feature space IS the input space.

Consequence. SVM with linear kernel = linear classifier. Boundary is a hyperplane in the input space.

When to use.

- Many features relative to samples (e.g., text classification with TF-IDF)
- Data is already roughly linearly separable
- Large datasets (faster to train than RBF)

Properties.

- No hyperparameter (other than C)
- Scales well: training is near-linear in n
- $\mathbf{w}$ can be written down explicitly: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

Library defaults. scikit-learn's `SVC(kernel='linear')` uses libsvm; for big data, prefer `LinearSVC` which scales better.

Linear kernel: identity transformation. Use it for high-dimensional or near-linearly-separable data.

Polynomial Kernel (Formula F9)

Formula F9.

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d.$$

Two hyperparameters.

- $d \in \{2, 3, 4, 5\}$: degree of the polynomial
- $c \geq 0$: constant that balances monomial weights

Monomial = a *product of powers* of input features, e.g., x_1^2 or $x_1 x_2$. *Degree* = total sum of exponents.

Implicit feature map. All monomial combinations of the input features up to degree d. For 2D input with $d = 2$: $\{1, x_1, x_2, x_1 x_2, x_1^2, x_2^2\}$.

When to use. Known polynomial structure. Feature interactions matter. Text data sometimes.

Warning.

- Large d creates an astronomically large implicit feature space
- Numerically unstable for $d > 5$ in practice
- Overfit risk: test performance often peaks at $d = 2$ or $d = 3$

Homogeneous vs inhomogeneous.

- $c = 0$: only degree-d monomials
- $c > 0$: monomials from degree 0 up to d (more flexible)

Default in scikit-learn: $d = 3$, $c = 1$.

Polynomial kernel: $(\mathbf{x} \cdot \mathbf{x}' + c)^d$. All monomials up to degree d. Use $d = 2$ or $d = 3$ in practice.

RBF Kernel (Formula F10)

Formula F10.

$$K(\mathbf{x}, \mathbf{x}') = \exp\bigl(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\bigr).$$

One hyperparameter. $\gamma > 0$ controls how fast similarity decays with distance.

Plain English.

- $\mathbf{x}$ and $\mathbf{x}'$ identical: $K = 1$ (maximal similarity)
- $\mathbf{x}$ and $\mathbf{x}'$ far apart: $K \approx 0$
- The "width" of the similarity bell is $1/\sqrt{\gamma}$

When to use. General-purpose nonlinear SVM. *The default.* If you do not know what kernel to use, start here.

Geometric intuition.

Each support vector $\mathbf{x}_i$ acts like a "similarity spotlight" centred on itself. The classification is a weighted sum of these spotlights, one per SV.

Implicit feature space. Infinite-dimensional — we would never compute $\varphi(\mathbf{x})$ explicitly. The kernel does all the work.

γ **rule of thumb.** Start with $\gamma = 1/(n_{\text{features}} \cdot \text{Var}(X))$ (scikit-learn's default: `gamma='scale'`).

RBF kernel: similarity decays with distance. γ sets the decay rate. Default choice for nonlinear SVM.

Worked Example 2: Evaluate Three Kernels

Test setup. Two 2D points:

$$\mathbf{x} = (1, 2), \quad \mathbf{x}' = (3, 1).$$

Task. Compute $K(\mathbf{x}, \mathbf{x}')$ for each kernel:

- Linear
- Polynomial with $d = 2$, $c = 1$
- RBF with $\gamma = 0.5$

Preview of answers.

- Linear: $K = 5$
- Polynomial: $K = 36$
- RBF: $K \approx 0.0821$

Same two points. Three very different similarity scores. The next three slides do the arithmetic.

Intermediate quantities we will reuse.

Dot product:

$$\mathbf{x} \cdot \mathbf{x}' = 1 \cdot 3 + 2 \cdot 1 = 3 + 2 = 5.$$

Squared distance:

$$\|\mathbf{x} - \mathbf{x}'\|^2 = (1 - 3)^2 + (2 - 1)^2 = 4 + 1 = 5.$$

These two numbers (5 and 5) feed into all three kernels. Keep them handy.

Two points, three kernels. We will compute each $K(\mathbf{x}, \mathbf{x}')$ from scratch on the next three slides.

Worked Example 2 Step 1: Linear Kernel

Formula. $K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$.

Plug in $\mathbf{x} = (1, 2)$, $\mathbf{x}' = (3, 1)$.

$$K = 1 \cdot 3 + 2 \cdot 1$$

$$= 3 + 2$$

$$= \mathbf{5}.$$

Interpretation. Two vectors with positive dot product are "aligned" in some sense. Larger K = more aligned.

Sanity check.

$\|\mathbf{x}\| = \sqrt{1^2 + 2^2} = \sqrt{5}$.
$\|\mathbf{x}'\| = \sqrt{3^2 + 1^2} = \sqrt{10}$.

$$\cos(\angle) = \frac{\mathbf{x} \cdot \mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|} = \frac{5}{\sqrt{5}\sqrt{10}} = \frac{5}{\sqrt{50}} \approx 0.707.$$

So the two vectors point in roughly similar but not identical directions (angle $\approx 45°$). Linear kernel reports this as "$K = 5$".

Linear kernel: $K = \mathbf{x} \cdot \mathbf{x}' = 5$. Just the raw dot product, no transformation.

Worked Example 2 Step 2: Polynomial Kernel

Formula. $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d$, $d = 2$, $c = 1$.

Reuse dot product. $\mathbf{x} \cdot \mathbf{x}' = 5$ (from Step 1).

Plug in.

$$K = (5 + 1)^2$$
$$= 6^2$$
$$= \mathbf{36}.$$

Interpretation. Polynomial kernel *amplifies* the dot product by raising it to a power. Aligned vectors $\Rightarrow$ very large K.

What implicit features did we buy?

For $d = 2$, $c = 1$, $\mathbf{x} = (x_1, x_2)$, the implicit feature map is

$$\varphi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2).$$

Six-dimensional feature space. SVM computes boundaries linear in φ, which correspond to quadratic boundaries in the original 2D space.

Sanity check.

$$\varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^2 = 36 \quad \checkmark$$

Polynomial kernel ($d = 2$, $c = 1$): $K = (5 + 1)^2 = 36$. Implicit feature space is 6-dimensional.

Worked Example 2 Step 3: RBF Kernel

Formula. $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$, $\gamma = 0.5$.

Compute squared distance.

$$\|\mathbf{x} - \mathbf{x}'\|^2 = (1-3)^2 + (2-1)^2$$

$$= 4 + 1 = 5.$$

Plug in.

$$K = \exp(-0.5 \cdot 5)$$

$$= \exp(-2.5)$$

$$\approx \mathbf{0.0821}.$$

Interpretation.

The two points are "medium far" in 2D. RBF reports this as a small similarity: $K \approx 0.08$.

Contrast with linear.

- Linear kernel: $K = 5$ (scalar alignment)
- RBF kernel: $K \approx 0.0821$ (bounded in $[0, 1]$, decays with distance)

Same raw data, three *qualitatively different* similarity scores:

- Linear: 5 (unbounded)
- Poly: 36 (amplifies)
- RBF: 0.0821 (dampens far apart)

RBF kernel ($\gamma = 0.5$): $K = \exp(-0.5 \cdot 5) = \exp(-2.5) \approx 0.0821$. **Decays with squared distance.**

Numeric Check: Polynomial Kernel $= \varphi \cdot \varphi'$

Verify $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^2 = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}')$ for an explicit φ.
Take $\mathbf{x} = (1, 2), \mathbf{x}' = (3, 1)$.

Direct kernel.

$$K = (\mathbf{x} \cdot \mathbf{x}' + 1)^2 = (1 \cdot 3 + 2 \cdot 1 + 1)^2 = 6^2 = 36.$$

Feature map:

$$\varphi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2).$$

So

$$\varphi(\mathbf{x}) = (1, \sqrt{2}, 2\sqrt{2}, 1, 4, 2\sqrt{2}),$$
$$\varphi(\mathbf{x}') = (1, 3\sqrt{2}, \sqrt{2}, 9, 1, 3\sqrt{2}).$$

Dot product in feature space.

$$\begin{aligned}\varphi(\mathbf{x}) \cdot \varphi(\mathbf{x}') &= 1 \cdot 1 \\ &\quad + \sqrt{2} \cdot 3\sqrt{2} \\ &\quad + 2\sqrt{2} \cdot \sqrt{2} \\ &\quad + 1 \cdot 9 \\ &\quad + 4 \cdot 1 \\ &\quad + 2\sqrt{2} \cdot 3\sqrt{2} \\ &= 1 + 6 + 4 + 9 + 4 + 12 \\ &= \mathbf{36}.\end{aligned}$$

Equal! $K = 36$ via the compact formula, $\varphi \cdot \varphi' = 36$ via explicit lifting. Both agree.

Moral. The kernel is a *shortcut*: it computes $\varphi \cdot \varphi'$ without ever building φ.

Direct: $K = 6^2 = 36$. **Explicit:** $\varphi \cdot \varphi' = 36$. **Matches. The kernel trick is bookkeeping, not magic.**

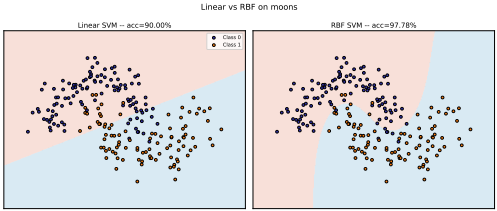
Linear vs RBF on Curved Data

Moons dataset. Two intertwined half-moons in 2D — the classic non-linearly-separable toy dataset.

Results.

- **Linear SVM**: test accuracy $\approx 90\%$. Forced to draw a straight line through curved data, so it cuts through each half-moon.
- **RBF SVM**: test accuracy $\approx 98\%$. The boundary bends around both half-moons cleanly.

Moral. When the true boundary is curved, the linear kernel hits a ceiling that no amount of data or C-tuning can raise.



Curved boundary needs curved classifier. Switching kernels from linear to RBF bought us 8 accuracy points.

The γ Hyperparameter

γ controls the RBF "bandwidth".

Recall $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$.

Two extremes.

- **Small γ** (wide bandwidth): each SV has broad influence. Smooth boundary, can underfit.
- **Large γ** (narrow bandwidth): each SV influences only nearby points. Tight local bumps, overfit risk.

Analogous to C.

- γ high $\Rightarrow$ low bias, high variance
- γ low $\Rightarrow$ high bias, low variance

Typical tuning strategy.

- Log-grid sweep: $\gamma \in \{10^{-3}, 10^{-2}, \ldots, 10^{1}\}$
- Tune jointly with C via cross-validation
- scikit-learn default: `gamma='scale'` which sets $\gamma = 1/(n_{\text{features}} \cdot \text{Var}(X))$

Rule of thumb. If the boundary is too spiky, reduce γ. If the boundary is too smooth and you are underfitting, increase γ.

γ sets RBF bandwidth. Small γ: smooth. Large γ: spiky. Tune on a log grid jointly with C.

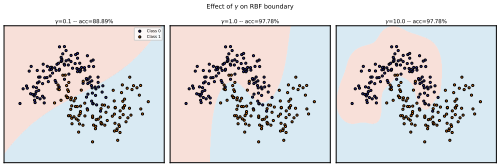
γ Sweep Visualised

Three γ values on the moons dataset.

- $\gamma = 0.1$: very smooth. Boundary barely bends; some misclassifications where the classes interlock.
- $\gamma = 1$: balanced. Boundary hugs both half-moons cleanly.
- $\gamma = 10$: tight bumps around each SV. Island-like decision regions. Overfits noise.

What to look for. Islands of wrong prediction in the middle panel are a warning sign of over-large γ.

Cross-validated answer: $\gamma \approx 1$ is usually best on this dataset.



Small γ: smooth underfit. Large γ: spiky overfit. Middle: the Goldilocks zone.

Hyperparameters Summary

Four knobs.

Knob	Range	Bias-Variance
kernel	lin/poly/rbf	lin: high bias
C	10^{-2} to 10^{2}	large: low bias
γ (RBF)	10^{-3} to 10^{1}	large: low bias
degree (poly)	2 to 5	large: low bias

Recommended workflow.

1. Start with RBF + `gamma='scale'` + $C = 1$
2. Sweep C on log grid first
3. Then sweep γ on log grid at best-C
4. Final pass: joint grid of (C, γ) if compute budget allows

Common pitfalls.

- Forgetting to *scale* features before training (RBF is distance-based)
- Tuning C without also tuning γ — they interact
- Using a linear kernel on curved data and blaming the penalty
- Using a very large d for the polynomial kernel (numerical instability)

Golden rule. Always do **feature standardisation** = rescale each feature to *mean 0* and variance 1 (std 1), so distances are comparable across features. Essential before training an SVM.

Four knobs. Tune (C, γ) jointly on a log grid. Always standardise features before training.

Cross-Validation in 1 Slide

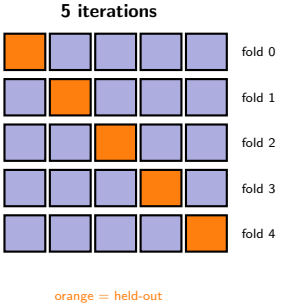
***k*-fold cross-validation.** Split the training data into k **folds** of equal size. For each fold:

1. Hold it out as a validation set
2. Train on the remaining $k-1$ folds
3. Measure accuracy on the **held-out fold**

Average the k accuracies to get a single honest estimate.

Concrete. 100 rows, $k = 5$ folds of 20 rows each. Train on 80, test on 20, *five times*. Each row is in the test set exactly once.

Use. Tune (C, γ) by running 5-fold CV for every grid point; pick the (C, γ) with highest average validation accuracy.



Default. $k = 5$ or $k = 10$. More folds $\Rightarrow$ more training data per fold $\Rightarrow$ less variance in the estimate but more compute.

5-fold CV = 5 train/test splits. Each row is held out exactly once. Averages over 5 accuracies.

Numeric Check: Complementary Slackness (Rebalance)

Re-check. On our 3-point setup with $\mathbf{w} = (1, 1), b = -1$, $\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 1$, verify each KKT product:

$$\alpha_i \cdot \left[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right] = 0 \quad \forall i.$$

Point 1 $(1, 1)$, $y = +1$, $\alpha = 1$:

- $y(\mathbf{w} \cdot \mathbf{x} + b) = 1 \cdot (1 + 1 - 1) = 1$
- bracket $= 1 - 1 = 0$
- product $= 1 \cdot 0 = 0$ ✓

Point 2 $(2, 2)$, $y = +1$, $\alpha = 0$:

- $y(\mathbf{w} \cdot \mathbf{x} + b) = 1 \cdot (2 + 2 - 1) = 3$
- bracket $= 3 - 1 = 2$
- product $= 0 \cdot 2 = 0$ ✓

Point 3 $(0, 0)$, $y = -1$, $\alpha = 1$:

- $y(\mathbf{w} \cdot \mathbf{x} + b) = (-1) \cdot (0 + 0 - 1) = 1$
- bracket $= 1 - 1 = 0$
- product $= 1 \cdot 0 = 0$ ✓

Summary. All three per-point products equal 0. This is *exactly* what F6 (and the KKT complementary-slackness condition) asserts.

What did we learn? The slackness identity is *per-point*: for each i either the α_i is 0, or the margin constraint is tight. Never both violated at once.

What it rules out. A training point deep past the margin can never have $\alpha_i > 0$ in a hard-margin SVM.

Per-point complementary slackness confirmed for all 3 points. SV condition is mechanical once α is known.

Numeric Check: Kernel Trick Preserves Dual Shape

Same 3 points: $(1,1), (2,2)$ $[+1]$; $(0,0)$ $[-1]$. Now apply the RBF kernel with $\gamma = 0.5$:

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-0.5\|\mathbf{u} - \mathbf{v}\|^2\right).$$

Kernel matrix.

$$K = \begin{pmatrix} 1 & e^{-1} & e^{-1} \\ e^{-1} & 1 & e^{-4} \\ e^{-1} & e^{-4} & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0.37 & 0.37 \\ 0.37 & 1 & 0.018 \\ 0.37 & 0.018 & 1 \end{pmatrix}.$$

Dual form unchanged.

$$\max_{\alpha} \sum_i \alpha_i - \tfrac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j K_{ij}.$$

Only the *matrix entries* swapped from $\mathbf{x}_i \cdot \mathbf{x}_j$ to K_{ij}.

What changes numerically.

- Linear case (old): $\alpha_1 = \alpha_3 = 1$, $\alpha_2 = 0$, dual value 1
- RBF case: the optimal α's shift (all three may be non-zero) to fit the new K, dual value changes

But the algorithmic shape is identical. Same QP solver, same constraints $0 \le \alpha_i \le C$ and $\sum_i \alpha_i y_i = 0$. Only the quadratic matrix changed.

That's the kernel trick. Change *one line* (the kernel function) and SVM becomes nonlinear without any other code change.

Prediction uses $f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$ — same formula, different kernel.

Swap $\mathbf{x}_i \cdot \mathbf{x}_j$ for K_{ij}. Dual form unchanged. Only the matrix entries (and hence α's) differ.

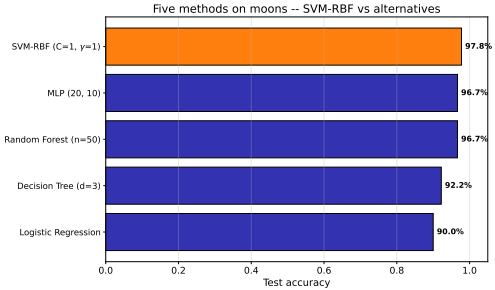
5-Method Comparison on Moons

Five classifiers, same dataset.

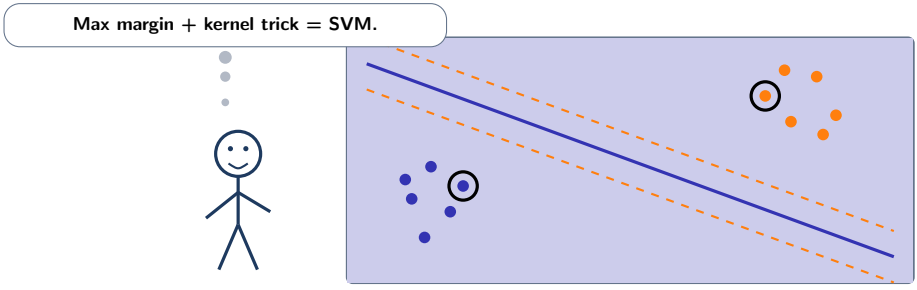
- Logistic Regression: $\approx 90\%$
- Decision Tree (depth 3): $\approx 92\%$
- Random Forest (50 trees): $\approx 97\%$
- MLP (20, 10): $\approx 97\%$
- **SVM-RBF** ($C = 1$, $\gamma = 1$): $\approx 98\%$

What the chart shows. On curved tabular data of modest size, SVM-RBF edges out tree ensembles and matches a neural network with a tiny fraction of the tuning effort.

Caveat. This is one dataset. Results vary. On very large data, or on structured inputs (images, text, sequences), deep models dominate.



Small curved tabular: SVM-RBF excels. On images, text, or very large data: pick other tools.



Support vectors circled. Margin widest subject to zero (soft) violations.