SUPPORT VECTOR MACHINES

FROM BEGINNER TO EXPERT

A complete, visual, application-driven course — from the maximal-margin idea through soft margins, the kernel trick, duality, and support vector regression, with a capstone on how SVMs and kernel methods power Machine Learning, Deep Learning, and AI.

What you will be able to do

Derive and train hard- and soft-margin SVMs; understand the margin, slack, and the hinge loss; master Lagrangian duality, the KKT conditions, and why support vectors appear; wield the kernel trick with polynomial and RBF kernels; tune C and γ; apply SVMs to regression, multiclass, and anomaly detection; and explain how large-margin and kernel ideas live on in modern deep learning and AI.

Format: self-paced reference & course | **Prerequisites:** linear algebra, a little calculus & probability

Part of the Comprehensive Machine Learning & Deep Learning Curriculum.

Preface: how to use this book

For most of the 1990s and 2000s, if you wanted the best off-the-shelf classifier, you reached for a Support Vector Machine. SVMs combined a beautiful geometric idea — separate the classes with the *widest possible street* — with deep optimization theory and the almost magical *kernel trick* that let a linear method draw curved boundaries in infinite-dimensional spaces. Deep learning later took the crown on perception tasks, but SVMs and the kernel viewpoint never left: they remain superb on small and medium data, and — remarkably — modern theory shows infinitely wide neural networks are themselves kernel machines. This book builds SVMs from the margin up to that frontier.

Who this is for

This book starts at the very beginning — what a margin is and why a wide one generalizes — and climbs to Lagrangian duality, reproducing-kernel Hilbert spaces, the SMO algorithm, and the SVM–neural-network connection. It is for the self-learner who wants both **fluency** ("I can train, tune, and deploy an SVM and know when to use one") and **understanding** ("I can derive the dual, explain support vectors via KKT, and see why the kernel trick works"). Every major idea gets a picture, a worked example, and a forward link to machine learning.

The central idea

All of SVM theory grows from one principle and its consequences:

- **Maximize the margin.** Among the infinitely many separating boundaries, choose the one farthest from the nearest points — the most robust one.
- **Only the hard cases matter.** The solution depends only on the boundary points — the *support vectors* — a sparsity that falls out of the KKT conditions.
- **Kernels buy nonlinearity for free.** Because everything depends on inner products, replacing them with a kernel draws nonlinear boundaries without ever visiting the high-dimensional space.

The structure

- **Part I — Foundations** (beginner): what SVMs are; the maximal-margin classifier; soft margins and slack.
- **Part II — The Mathematics** (core): hinge loss and the primal; Lagrangian duality and KKT; the kernel trick.
- **Part III — Variants & Algorithms** (advanced): kernels in depth; support vector regression; how SVMs are solved.
- **Part IV — Practice & Theory** (advanced/expert): multiclass, probabilities, and practical SVMs; margins, VC dimension, and generalization.
- **Part V — Frontier**: SVMs versus the world; and a capstone on SVMs and kernels in ML/DL/AI.

How to read it

Read with a pen and a computer. When you meet a *Definition*, restate it; when you meet an *Example*, work it before reading on; when you meet a *Try it in code* box, run it. The colored boxes recur throughout:

- **Key idea** — the one sentence to remember.
- **Intuition** — the picture or analogy behind the math.
- **Common pitfall** — the mistakes that bite everyone.
- **How this powers ML / AI** — the forward link to applications.
- **Try it in code** — a hands-on check in Python (scikit-learn).

Key idea

Hold one picture above all others: an SVM finds the **widest possible street** between two classes; the points touching the curb are the **support vectors**, and they alone define the boundary. Everything else — soft margins, the dual problem, KKT, the kernel trick — is machinery that makes this idea robust to noise and powerful enough for nonlinear data. Master the margin, and support vector regression, kernels, and the deep-learning connection are the same idea, extended.

Contents

PART I

Foundations

1

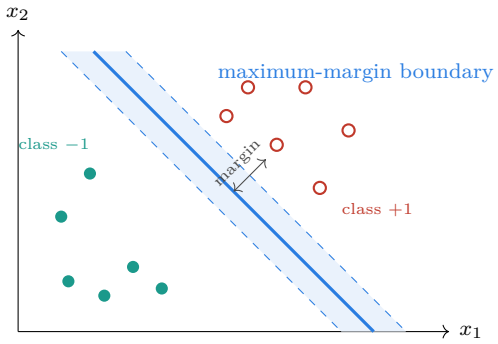
What Are Support Vector Machines?

Level: Beginner

Imagine two groups of points on a page and a ruler you must lay down to separate them. Many positions work — but which is *best*? A Support Vector Machine answers: the line that leaves the **widest empty street** between the groups. That single instinct, made precise and extended with the kernel trick, gives one of the most elegant and powerful classifiers ever invented. This chapter sets up the idea, the vocabulary, and why it matters.

1.1 The widest-street idea

Definition 1.1. A **Support Vector Machine (SVM)** is a supervised model that separates two classes with the **maximum-margin** decision boundary — the hyperplane that is as far as possible from the nearest points of either class. The nearest points, which "hold up" the boundary, are the **support vectors**.



Intuition

Why the widest street? A boundary crammed against the data is fragile — a tiny wiggle in a point flips a prediction. A boundary with a wide buffer is *robust*: new points have to move a lot before they cross. Maximizing the margin is a built-in safety factor, and (Chapter 11) it is exactly what controls how well the model generalizes to unseen data. The SVM does not just separate; it separates with maximum confidence.

1.2 The decision rule

Key idea

An SVM classifies with a linear score and its sign:

$$\hat{y} = \operatorname{sgn}\big(\mathbf{w}^\top \mathbf{x} + b\big),$$

where $\mathbf{w}$ is the weight vector (perpendicular to the boundary) and b is the offset. The boundary is the set $\mathbf{w}^\top \mathbf{x} + b = 0$; the margins are the parallel surfaces $\mathbf{w}^\top \mathbf{x} + b = \pm 1$. Training chooses $\mathbf{w}, b$ to make the street between those margins as wide as possible while keeping points on the correct side.

1.3 Three ideas that make SVMs special

- **Maximum margin** (Chapters 2, 3): choose the most robust boundary; allow a few violations with a budget C.
- **Support vectors** (Chapter 5): the solution depends only on the handful of boundary points — a sparsity that falls out of the optimization (KKT conditions).
- **The kernel trick** (Chapter 6): because everything depends on inner products $\mathbf{x}_i^\top \mathbf{x}_j$, replacing them with a *kernel* $k(\mathbf{x}_i, \mathbf{x}_j)$ draws nonlinear boundaries in a high- (even infinite-) dimensional space without ever computing coordinates there.

1.4 A short history

Intuition

The linear maximal-margin idea goes back to Vapnik and Chervonenkis in the 1960s; the modern SVM — soft margins plus the kernel trick — was introduced by **Cortes and Vapnik (1995)**, building on Boser, Guyon, and Vapnik (1992). Through the late 1990s and 2000s SVMs were the dominant "off-the-shelf" classifier, winning at text categorization, handwriting and digit recognition, and bioinformatics. Deep learning overtook them on large perceptual datasets after 2012, but SVMs remain a first-class tool for small/medium structured data — and their theory underpins modern results connecting kernels to neural networks.

How this powers ML / AI

SVMs gave machine learning several durable gifts. The **hinge loss** they minimize reappears as a standard classification loss in deep networks. The **large-margin principle** is the ancestor of margin-based and contrastive losses used to train embeddings and face-recognition models. The **kernel viewpoint** matured into a whole field (Gaussian processes, kernel ridge regression) and, strikingly, modern theory shows an infinitely wide neural network behaves like a kernel machine — a Support Vector Machine with the Neural Tangent Kernel (Chapter 13). Learning SVMs is learning the grammar of much of modern ML.

Try it in NumPy

```
from sklearn.svm import SVC
from sklearn.datasets import load_iris
```

```
X, y = load_iris(return_X_y=True)
X, y = X[y < 2, :2], y[y < 2]            # two classes, two features
clf = SVC(kernel="linear", C=1.0).fit(X, y)
print("accuracy:", clf.score(X, y))
print("number of support vectors per class:", clf.n_support_)
print("w =", clf.coef_.round(2), " b =", round(clf.intercept_[0], 2))
```

Chapter summary

- An **SVM** separates classes with the **maximum-margin** hyperplane — the widest empty street between them.
- The decision rule is $\hat{y} = \operatorname{sgn}(\mathbf{w}^\top \mathbf{x} + b)$; margins are the surfaces $\mathbf{w}^\top \mathbf{x} + b = \pm 1$.
- Only the boundary points — the **support vectors** — determine the solution.
- Three pillars: **maximum margin**, **support-vector sparsity**, and the **kernel trick** for nonlinearity.
- Introduced by Cortes & Vapnik (1995); still excellent on small/medium data and foundational to modern ML theory.

2

The Maximal Margin Classifier

Level: Beginner

Now we make the widest-street idea precise. For data that *can* be perfectly separated by a line, the maximal margin classifier (the hard-margin SVM) is the cleanest version of the story: a little geometry turns "make the street as wide as possible" into a concrete optimization problem whose solution depends only on the support vectors.

2.1 Hyperplanes and the geometry of the margin

Definition 2.1. A **hyperplane** in $\mathbb{R}^p$ is the set $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = 0\}$. The vector $\mathbf{w}$ is **normal** (perpendicular) to it. The **signed distance** from a point $\mathbf{x}_0$ to the hyperplane is

$$\frac{\mathbf{w}^\top \mathbf{x}_0 + b}{\|\mathbf{w}\|}.$$

Intuition

The sign of $\mathbf{w}^\top \mathbf{x} + b$ says which side of the boundary you are on; the magnitude (divided by $\|\mathbf{w}\|$) says how far. So if we label classes $y \in \{-1, +1\}$, the quantity $y_i(\mathbf{w}^\top \mathbf{x}_i + b)$ is positive exactly when point i is on the correct side, and its size measures the margin of that point. This is why the labels are chosen as ± 1 rather than $0/1$ — it makes "correct and confident" a single clean inequality.

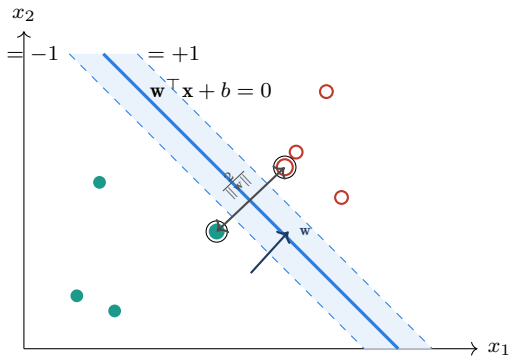
2.2 The canonical form and the width of the street

Key idea

We are free to rescale $\mathbf{w}, b$, so we fix the scale by requiring the closest points to satisfy $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$. The two margin surfaces are then $\mathbf{w}^\top \mathbf{x} + b = \pm 1$, and the distance between them — the **width of the street** — works out to

$$\text{margin} = \frac{2}{\|\mathbf{w}\|}.$$

Maximizing the margin is therefore the same as *minimizing* $\|\mathbf{w}\|$ (equivalently $\frac{1}{2}\|\mathbf{w}\|^2$).



2.3 The hard-margin optimization problem

Definition 2.2. The **maximal margin (hard-margin) classifier** solves

$$\min_{\mathbf{w},\, b} \ \tfrac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \ \geq \ 1 \quad \text{for all } i.$$

This is a **convex quadratic program**: a quadratic objective with linear constraints, so it has a unique global optimum.

Intuition

Read the problem in plain words: "make the weight vector as short as possible (widest street) while keeping every point at least one full margin onto its correct side." Convexity matters enormously — unlike training a neural network, there are no local minima or random restarts; the same data always yields the same optimal boundary. The solution turns out to be a weighted combination of just the support vectors (Chapter 5).

2.4 Support vectors

Key idea

At the optimum, points fall into two groups. **Support vectors** sit exactly on the margin ($y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$) and *determine* the boundary. All other points are strictly beyond the margin and are *irrelevant*: delete them and re-train, and you get the same boundary. This is the SVM's defining sparsity — the model is summarized by a few critical examples, not the whole dataset.

Common pitfall

The hard-margin classifier has two fatal flaws on real data: (1) it **requires perfect linear separability** — if even one point is on the wrong side, the constraints are infeasible and there is *no* solution; and (2) it is **exquisitely sensitive to outliers** — a single mislabeled point can swing the boundary wildly or shrink the margin to nothing. Real data is noisy and overlapping, so we almost never use hard margins directly. The fix — allowing a budget of violations — is the soft-margin SVM of the next chapter.

How this powers ML / AI

The move "maximize margin $\Leftrightarrow$ minimize $\|\mathbf{w}\|^2$" is the first appearance of L_2 **regularization** in this book, and it is no coincidence: penalizing $\|\mathbf{w}\|^2$ is exactly weight decay in neural networks and ridge regression in statistics. The SVM reveals the geometric meaning of that penalty — shorter weights mean wider margins mean more robust decisions. The same principle, "prefer the flattest function that fits," echoes through every regularized model in modern AI.

Try it in NumPy

```python
import numpy as np
from sklearn.svm import SVC
# Perfectly separable toy data
X = np.array([[1,1],[2,1],[1,2],[5,5],[6,5],[5,6]])
y = np.array([-1,-1,-1, 1, 1, 1])
clf = SVC(kernel="linear", C=1e6).fit(X, y)        # huge C ~ hard
    margin
w, b = clf.coef_[0], clf.intercept_[0]
print("margin width 2/||w|| =", round(2/np.linalg.norm(w), 3))
print("support vectors:\n", clf.support_vectors_)   # only the
    boundary points
```

Chapter summary

- A hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ has normal $\mathbf{w}$; the signed distance of a point is $(\mathbf{w}^\top \mathbf{x} + b)/\|\mathbf{w}\|$.
- With labels $y \in \{\pm 1\}$ and the canonical scaling, the margin width is $2/\|\mathbf{w}\|$.
- **Maximizing the margin $\Leftrightarrow$ minimizing** $\frac{1}{2}\|\mathbf{w}\|^2$ subject to $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$ — a convex QP.
- Only **support vectors** (points on the margin) determine the boundary; the rest are irrelevant.
- Hard margins fail on non-separable or noisy data and are outlier-sensitive — motivating soft margins next.

3

Soft Margins and Slack

Level: Core

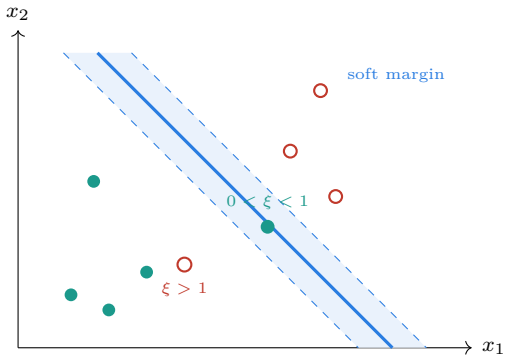
Real data overlaps, contains noise, and is rarely perfectly separable. The **soft-margin SVM** — the model people actually use — relaxes the hard constraints by allowing a controlled budget of margin violations. This single change makes SVMs robust, introduces the all-important hyperparameter C, and reframes the SVM as a regularized loss-minimization problem.

3.1 Slack variables

Definition 3.1. The **soft-margin SVM** introduces a **slack variable** $\xi_i \geq 0$ for each point, measuring how far it violates its margin, and solves

$$\min_{\mathbf{w},b,\xi} \ \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$

Here $\xi_i = 0$ means point i is safely on or beyond its margin; $0 < \xi_i < 1$ means inside the street but correctly classified; $\xi_i > 1$ means misclassified.



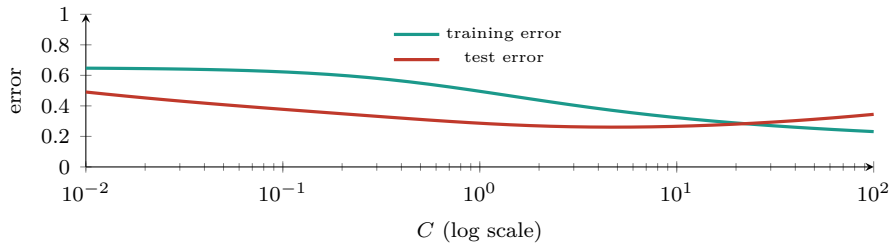
3.2 The role of C

Key idea

The hyperparameter $C > 0$ sets the **price of a violation** — it trades margin width against training errors:

- **Large** C: violations are expensive ⇒ the model tolerates few errors, narrow margin, fits training data hard — *low bias, high variance* (toward hard margin).
- **Small** C: violations are cheap ⇒ the model accepts more errors for a wider, smoother

margin — *high bias, low variance* (more regularized).
C is the SVM's primary regularization knob and is tuned by cross-validation.



3.3 More support vectors

Intuition

In the soft-margin world, support vectors are all points with $\xi_i > 0$ *or* sitting exactly on the margin — i.e., everything inside or touching the street, plus the misclassified points. Smaller $C \Rightarrow$ wider street $\Rightarrow$ *more* support vectors $\Rightarrow$ a more stable, more regularized model. The fraction of points that become support vectors is a useful diagnostic: very high means the problem is hard or C is small; very low means an easy, well-separated problem.

3.4 The two faces of the objective

Key idea

Rearranging the constraints, the optimal slack is $\xi_i = \max(0,\, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$ — the **hinge loss**. So the soft-margin SVM is exactly

$$\min_{\mathbf{w},b} \underbrace{\tfrac{1}{2}\|\mathbf{w}\|^2}_{\text{margin / regularizer}} + C \sum_i \underbrace{\max(0,\, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))}_{\text{hinge loss (data fit)}}.$$

This is the **regularized-loss view**: SVM = L_2 regularization + hinge loss. It connects SVMs to logistic regression (swap hinge for log-loss) and to all of modern ML, and it is the launchpad for the next chapter.

Common pitfall

Two recurring mistakes. (1) **Forgetting to scale features.** The penalty $\|\mathbf{w}\|^2$ and distances depend on units, so a feature measured in thousands dominates one measured in fractions. *Always standardize* (zero mean, unit variance) before training an SVM. (2) **Misreading C.** In scikit-learn, *larger* C means *less* regularization (opposite to the λ in ridge regression) — a frequent source of backwards tuning. Sweep C on a log grid and let cross-validation decide.

How this powers ML / AI

The regularized-loss form "$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum$ loss" is the template of essentially every trained model in machine learning, deep learning included: a complexity penalty plus a data-fitting

loss, balanced by a coefficient. Recognizing the SVM as one instance of this template — hinge loss + L_2 — makes the whole landscape legible: logistic regression is log-loss + L_2, a neural net is cross-entropy + weight decay, and so on. C is the same dial as weight-decay strength, just inverted.

Try it in NumPy

```python
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
X, y = make_classification(n_samples=400, n_features=10, class_sep=0.8, random_state=0)
for C in [0.01, 0.1, 1, 10, 100]:
    clf = make_pipeline(StandardScaler(), SVC(kernel="linear", C=C))  # scale first!
    print(f"C={C:>6}: cv acc={cross_val_score(clf, X, y, cv=5).mean():.3f}")
```

Chapter summary

- The **soft-margin SVM** adds **slack** $\xi_i \geq 0$ to permit margin violations: $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$.
- The objective $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum \xi_i$ trades margin width against violations via C.
- **Large** $C \to$ narrow margin, low bias/high variance; **small** $C \to$ wide margin, more regularized.
- Optimal slack is the **hinge loss**, so SVM = L_2 regularization + hinge loss (the regularized-loss view).
- **Always standardize features**; remember scikit-learn's C is *inverse* regularization.

PART II

The Mathematics

4

Hinge Loss and the Primal Problem

Level: Core

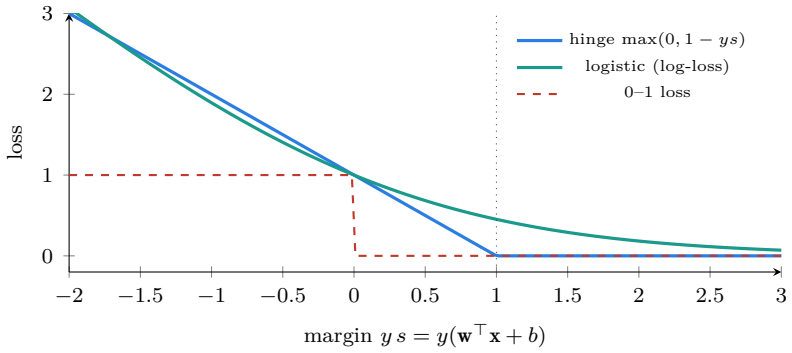
We ended the last chapter with a striking reformulation: the soft-margin SVM is just L_2 regularization plus a special loss. That loss is the **hinge loss**, and viewing the SVM as "minimize regularized hinge loss" (the *primal* problem) demystifies it, connects it to logistic regression, and lets us train SVMs by plain gradient descent — the same way we train neural networks.

4.1 The hinge loss

Definition 4.1. The **hinge loss** for a labeled point $(\mathbf{x}, y)$ with score $s = \mathbf{w}^\top \mathbf{x} + b$ is

$$\ell_{\text{hinge}}(y, s) = \max(0,\ 1 - y\, s).$$

It is zero once the point is correct *with margin* ($y\, s \geq 1$), and grows linearly as the point moves into or across the margin.



Intuition

The hinge loss is a **convex surrogate** for the (discontinuous, intractable) 0–1 misclassification loss. Crucially, it has a *flat zero region*: points already classified with enough margin contribute *nothing* to the loss or its gradient. That flat region is precisely why SVMs are sparse — only points on or inside the margin (the support vectors) push on the solution. Compare logistic loss, which is always positive: every point nudges a logistic model, so it has no support vectors.

4.2 The primal objective

Key idea

The (soft-margin) SVM **primal problem** is unconstrained minimization of regularized hinge loss:

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

Equivalently, dividing by C, it is $\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_i \ell_{\text{hinge}}$ with $\lambda \propto 1/C$ — average hinge loss plus an L_2 penalty. This is a convex, piecewise-quadratic function of $(\mathbf{w}, b)$ with a unique minimum.

4.3 Training by (sub)gradient descent

Intuition

Because the primal is convex, we can minimize it directly. The hinge loss has a kink at $ys = 1$, so we use a **subgradient**:

$$\nabla_{\mathbf{w}} \ell_i = \begin{cases} \mathbf{0} & \text{if } y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \text{(safe point)}, \\ -\,y_i\,\mathbf{x}_i & \text{otherwise} \quad \text{(margin violator)}. \end{cases}$$

The full gradient adds the regularizer's $\mathbf{w}$. A point only contributes a "push" $(-y_i\mathbf{x}_i)$ when it violates its margin — the update literally drags the boundary toward the troublemakers. Stochastic subgradient descent (e.g. the *Pegasos* algorithm) trains linear SVMs on huge datasets this way.

4.4 Hinge versus logistic

	SVM (hinge)	**Logistic regression (log-loss)**
Loss	$\max(0, 1 - ys)$	$\log(1 + e^{-ys})$
Zero-loss region	yes ($ys \geq 1$)	no (always > 0)
Solution sparsity	support vectors only	uses all points
Probabilities	not native (needs calibration)	native $\sigma(s)$
Robust to outliers	moderately (linear growth)	similar (linear tail)

Common pitfall

Don't expect probabilities from a raw SVM. The hinge loss optimizes the *decision boundary*, not calibrated likelihoods; the score $\mathbf{w}^\top\mathbf{x} + b$ is a signed distance, not $P(y \mid \mathbf{x})$. If you need probabilities, fit a calibration map (Platt scaling / isotonic) on held-out data — which is what scikit-learn's `probability=True` does internally (Chapter 10). Also note the kink: standard gradient descent works fine with subgradients, but "the gradient is undefined at $ys = 1$" trips up naive implementations.

How this powers ML / AI

The hinge loss is not a museum piece — it is a standard tool in deep learning. "Linear-SVM-on-top" networks replace the softmax/cross-entropy head with a **linear hinge (L2-SVM) loss** and sometimes train *better* on classification benchmarks. Margin-based and triplet losses for metric learning, face recognition, and retrieval are direct descendants of the hinge's "correct by a margin or pay" principle. And the primal view — regularizer plus convex surrogate loss, minimized by (stochastic sub)gradient descent — is *exactly* how neural networks are trained, making the SVM the cleanest gateway to understanding modern optimization.

Try it in NumPy

```python
import numpy as np
from sklearn.linear_model import SGDClassifier   # linear SVM via subgradient descent
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
X, y = make_classification(n_samples=2000, n_features=20, random_state=0)
clf = make_pipeline(StandardScaler(),
                    SGDClassifier(loss="hinge", alpha=1e-3, max_iter=1000))  # hinge = SVM
clf.fit(X, y)
print("train accuracy:", round(clf.score(X, y), 3))
```

Chapter summary

- The **hinge loss** $\max(0, 1 - ys)$ is a convex surrogate for 0–1 loss with a **flat zero region** beyond the margin.
- The SVM **primal** minimizes $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \text{hinge}_i$ — L_2 regularization plus average hinge loss.
- The flat region makes the solution **sparse** (support vectors); logistic loss has no such region.
- The primal is convex and trainable by **(sub)gradient descent** (e.g. Pegasos), scaling to large linear problems.
- SVMs don't output probabilities natively; **calibrate** if needed. Hinge/margin losses live on throughout deep learning.

5

Lagrangian Duality and the KKT Conditions

Level: Advanced

The primal view trains SVMs by gradient descent, but it hides two treasures: *why* support vectors appear, and *how* the kernel trick becomes possible. Both emerge when we rewrite the SVM through **Lagrangian duality**. This is the most mathematical chapter of the book; take it slowly, because the dual form is the gateway to everything nonlinear.

5.1 Constrained optimization and the Lagrangian

Definition 5.1. For the hard-margin problem $\min \frac{1}{2}\|\mathbf{w}\|^2$ s.t. $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$, introduce a **Lagrange multiplier** $\alpha_i \geq 0$ for each constraint and form the **Lagrangian**

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \big[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1\big].$$

Intuition

A Lagrange multiplier is the "price" of a constraint. Each α_i measures how hard constraint i is pushing on the solution. The art of duality: minimize $\mathcal{L}$ over the primal variables $(\mathbf{w}, b)$ and *maximize* over the prices α. Under convexity (which the SVM enjoys), this min–max can be swapped (*strong duality*), and solving the resulting "dual" problem in α recovers the exact primal solution.

5.2 Deriving the dual

Key idea

Setting $\partial\mathcal{L}/\partial\mathbf{w} = 0$ and $\partial\mathcal{L}/\partial b = 0$ gives the **stationarity conditions**

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \qquad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

The first is profound: **the optimal weight vector is a linear combination of the training points**, weighted by $\alpha_i y_i$. Substituting back eliminates $\mathbf{w}, b$ and yields the **dual problem**:

$$\max_{\alpha \geq 0} \; \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j \, y_i y_j \, \mathbf{x}_i^\top \mathbf{x}_j \quad \text{s.t.} \quad \sum_i \alpha_i y_i = 0, \quad (0 \leq \alpha_i \leq C).$$

The box constraint $\alpha_i \le C$ appears in the soft-margin case.

Intuition

Stare at the dual objective: the data enters *only* through inner products $\mathbf{x}_i^\top \mathbf{x}_j$. The features themselves have vanished — we never need the coordinates of $\mathbf{x}_i$, only their pairwise similarities. *This* is the doorway to the kernel trick (Chapter 6): replace $\mathbf{x}_i^\top \mathbf{x}_j$ with any valid kernel $k(\mathbf{x}_i, \mathbf{x}_j)$ and you are solving an SVM in a high-dimensional feature space, at no extra cost. Duality is not just elegant; it is what makes nonlinear SVMs computable.

5.3 KKT conditions and the birth of support vectors

Key idea

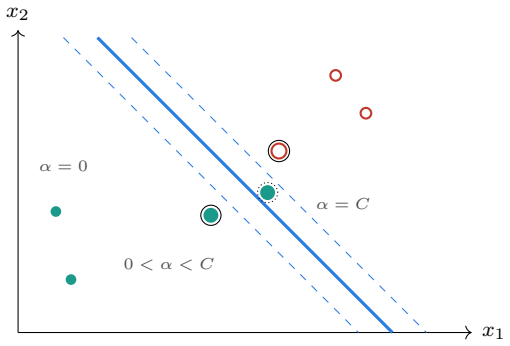
At the optimum, the **Karush–Kuhn–Tucker (KKT)** conditions hold; the decisive one is **complementary slackness**:

$$\alpha_i \big[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i \big] = 0.$$

This forces an either/or for every point:

- $\alpha_i = 0$: the point is strictly beyond the margin — *not* a support vector, no influence.
- $0 < \alpha_i < C$: the point sits exactly on the margin — a *free support vector*.
- $\alpha_i = C$: the point violates its margin ($\xi_i > 0$) — a *bounded support vector*.

So the support vectors are exactly the points with $\alpha_i > 0$, and the prediction $\hat{y} = \operatorname{sgn}(\sum_i \alpha_i y_i \, \mathbf{x}_i^\top \mathbf{x} + b)$ sums over *them alone*.



5.4 Why the dual matters

Intuition

Three payoffs. (1) **Kernels:** the dual depends only on inner products, enabling nonlinear SVMs. (2) **Sparsity:** most $\alpha_i = 0$, so prediction is cheap — only support vectors are stored. (3) **Insight:** the multipliers α_i rank how "critical" each example is. The cost: the dual is an $n \times n$ quadratic program (the *Gram matrix* of all pairwise kernels), so it scales poorly with the number of *samples* — which is why huge linear problems are often solved in the primal instead (Chapter 9).

Common pitfall

Primal or dual? Use the **dual** when you need kernels or when p (features) $\gg n$ (samples). Use the **primal** (subgradient/Pegasos, LIBLINEAR) for linear SVMs when n is large, since the dual's $n \times n$ Gram matrix becomes intractable past $\sim 10^5$ points. A common mistake is reaching for a kernel SVM on a million rows — it will crawl; either subsample, use a linear SVM, or use an explicit feature approximation (random Fourier features, Chapter 7).

How this powers ML / AI

Lagrangian duality and KKT are not SVM-specific — they are the backbone of constrained optimization across AI: they underlie **support-vector sparsity**, the dual formulations used in **reinforcement learning** (constrained policy optimization), **optimal transport**, and **adversarial robustness** (min–max games). The interpretation of multipliers as sample importance recurs in modern theory: in the SVM–neural-network correspondence, the Lagrange multipliers of the equivalent SVM measure which training points most shape an infinitely wide network (Chapter 13). Mastering duality here pays dividends far beyond SVMs.

Try it in NumPy

```
import numpy as np
from sklearn.svm import SVC
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=40, centers=2, random_state=6)
clf = SVC(kernel="linear", C=1.0).fit(X, y)
print("dual coefs (alpha*y) for SVs:", clf.dual_coef_.round(2))
print("# support vectors:", len(clf.support_), "of", len(X))
# reconstruct w from the dual: w = sum_i alpha_i y_i x_i  (over
    support vectors)
w = (clf.dual_coef_ @ clf.support_vectors_).ravel()
print("w from dual:", w.round(2), " vs clf.coef_:", clf.coef_.ravel()
    .round(2))
```

Chapter summary

- The **Lagrangian** attaches multipliers $\alpha_i \geq 0$ to the margin constraints; convexity gives **strong duality**.
- Stationarity yields $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$ and $\sum_i \alpha_i y_i = 0$ — the weight vector is a combination of training points.
- The **dual** depends on data only through inner products $\mathbf{x}_i^\top \mathbf{x}_j$ — the doorway to the **kernel trick**.
- **KKT complementary slackness** makes $\alpha_i > 0$ only for **support vectors**; free ($0 < \alpha < C$) vs bounded ($\alpha = C$).
- Dual for kernels / $p \gg n$; primal for large n (the dual's $n \times n$ Gram matrix limits scale).

6

The Kernel Trick

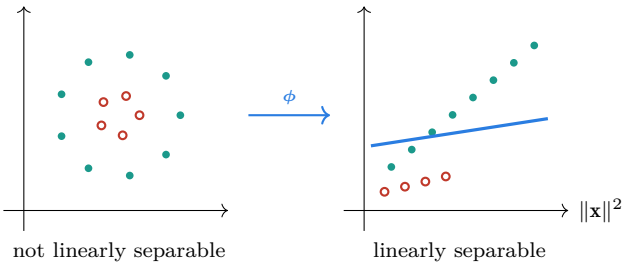
Level: Advanced

Here is the idea that turned a linear classifier into one of the most powerful tools in machine learning. A linear SVM can only draw straight boundaries — useless for data shaped like rings or spirals. The **kernel trick** lets the very same algorithm draw arbitrarily curved boundaries, by computing inner products in a high-dimensional space *without ever going there*. It is built directly on the dual form of the previous chapter.

6.1 Lifting data into feature space

Key idea

If data is not linearly separable in its original space, map it with a **feature map** $\boldsymbol{\phi} : \mathbb{R}^p \to \mathbb{R}^D$ (with D large) into a space where it *is* linearly separable, then run a linear SVM there. The boundary is linear in the lifted space but **nonlinear** back in the original space.



6.2 The trick: kernels replace inner products

Definition 6.1. A **kernel** is a function $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$ that computes the inner product in feature space *directly from the inputs*, without forming $\boldsymbol{\phi}$. The **kernel trick** is the observation that the SVM dual and prediction use data only through inner products, so we may replace every $\mathbf{x}_i^\top \mathbf{x}_j$ with $k(\mathbf{x}_i, \mathbf{x}_j)$.

Example 6.2. Let $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$ in $\mathbb{R}^2$. Expanding,

$$(x_1 x_1' + x_2 x_2')^2 = (x_1^2)(x_1'^2) + (x_2^2)(x_2'^2) + 2(x_1 x_2)(x_1' x_2'),$$

which is the inner product of $\boldsymbol{\phi}(\mathbf{x}) = (x_1^2,\ x_2^2,\ \sqrt{2}\, x_1 x_2)$ with $\boldsymbol{\phi}(\mathbf{x}')$. So computing one scalar product and squaring it secretly evaluates a dot product in a 3-D feature space — and for degree d in p dimensions, in a space of dimension $\binom{p+d}{d}$, which can be astronomically large.

The RBF kernel corresponds to an *infinite*-dimensional feature space.

6.3 The kernelized SVM

Key idea

Replacing inner products with k, the dual becomes $\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j\, k(\mathbf{x}_i, \mathbf{x}_j)$, and prediction is

$$\hat{y}(\mathbf{x}) = \operatorname{sgn}\Big(\sum_{i\in \mathrm{SV}} \alpha_i y_i\, k(\mathbf{x}_i, \mathbf{x}) + b\Big).$$

The model never stores $\mathbf{w}$ (it may be infinite-dimensional) — it stores the support vectors and their α_i, and classifies new points by comparing them, via k, to the support vectors.

6.4 What makes a valid kernel?

Definition 6.3. By **Mercer's theorem**, k is a valid kernel (corresponds to *some* feature map) if and only if it is symmetric and **positive semi-definite**: for any points, the **Gram matrix** $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ has no negative eigenvalues. Such kernels define a **Reproducing Kernel Hilbert Space (RKHS)** — the (possibly infinite-dimensional) feature space in which the SVM is linear.

Intuition

Think of a kernel as a **similarity measure**: $k(\mathbf{x}, \mathbf{x}')$ is large when $\mathbf{x}, \mathbf{x}'$ are "alike" in the relevant sense. The positive-semidefinite requirement is what guarantees this similarity is a genuine geometry (an inner product), so the optimization stays convex. A bonus of the RKHS view is the **representer theorem**: the optimal function is always a finite sum $\sum_i \alpha_i k(\mathbf{x}_i, \cdot)$ over the training points — even in infinite dimensions, the solution lives in the span of the data.

Common pitfall

The kernel trick is powerful enough to overfit spectacularly. An RBF kernel with too-large γ (tiny bandwidth) makes every point its own island — perfect training accuracy, useless generalization. Other traps: **forgetting to standardize** (RBF distances become meaningless across mismatched scales); using a **non-PSD "kernel"** (e.g. a tuned sigmoid kernel), which breaks convexity and may not converge; and applying kernels to **huge** n, where the $n\times n$ Gram matrix exhausts memory. Tune C and the kernel parameters together by cross-validation (Chapters 7, 10).

How this powers ML / AI

The kernel trick launched an entire paradigm — **kernel methods** — spanning Gaussian processes, kernel ridge regression, kernel PCA, and one-class SVMs. Its modern echo is the deepest result connecting SVMs to deep learning: an infinitely wide neural network is governed by the **Neural Tangent Kernel**, so training such a network by a margin loss is provably equivalent to a *kernel SVM* (Chapter 13). The "compute similarities, not coordinates" idea also prefigures **attention** in transformers, where outputs depend on

learned similarities (dot products) between tokens. Kernels are everywhere once you know to look.

Try it in NumPy

```python
import numpy as np
from sklearn.svm import SVC
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=300, factor=0.4, noise=0.1,
    random_state=0)
lin = SVC(kernel="linear").fit(X, y)
rbf = SVC(kernel="rbf", gamma=1.0, C=1.0).fit(X, y)
print("linear kernel accuracy:", round(lin.score(X, y), 3))   # ~0.5,
    can't separate rings
print("RBF kernel accuracy:   ", round(rbf.score(X, y), 3))   # ~1.0,
    curved boundary
```

Chapter summary

- A **feature map** ϕ lifts data to a space where it is linearly separable; the boundary is nonlinear back home.
- A **kernel** $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$ computes feature-space inner products without forming ϕ.
- The kernelized prediction $\operatorname{sgn}(\sum_{\text{SV}} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b)$ uses only support vectors and the kernel.
- Valid kernels are symmetric **positive semi-definite** (Mercer); they define an **RKHS**, and the **representer theorem** keeps solutions finite.
- Kernels can overfit (watch γ), need scaled features, and scale poorly with n; they seeded all of kernel methods and the NTK link to deep nets.

PART III

Variants and Algorithms

7

Kernels in Depth

Level: Advanced

The kernel is where you inject knowledge about your data's structure. Choosing and tuning it is the single most consequential decision in applying an SVM. This chapter surveys the standard kernels, explains the crucial γ parameter of the RBF kernel, shows how kernels extend SVMs to strings, graphs, and other non-vector data, and addresses how to scale kernels to large datasets.

7.1 The standard kernels

Kernel	**Formula** $k(\mathbf{x}, \mathbf{x}')$	**Use / notes**
Linear	$\mathbf{x}^\top \mathbf{x}'$	high-dim sparse data (text); fast
Polynomial	$(\gamma\, \mathbf{x}^\top \mathbf{x}' + r)^d$	explicit feature interactions up to degree d
RBF / Gaussian	$\exp(-\gamma \lVert \mathbf{x} - \mathbf{x}' \rVert^2)$	default nonlinear kernel; local, smooth
Sigmoid	$\tanh(\gamma\, \mathbf{x}^\top \mathbf{x}' + r)$	neural-net-like; *not* always PSD

Key idea

The **RBF (Gaussian) kernel** is the default first choice for nonlinear problems. It measures similarity as a bump that decays with distance: nearby points are similar ($k \to 1$), far points dissimilar ($k \to 0$). It can approximate almost any decision boundary, has only one shape parameter γ, and corresponds to an infinite-dimensional feature space. Start with RBF unless you have a reason (linear for text/very-high-dim; polynomial for known interaction structure; custom for structured objects).

7.2 The bandwidth γ

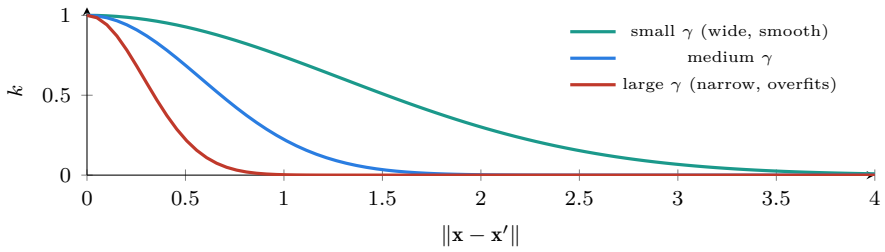
Intuition

For the RBF kernel, γ controls the **reach** of each support vector — it is an inverse bandwidth:

- **Small** γ (wide bumps): each point influences a large region $\Rightarrow$ smooth, simple boundary,

can *underfit*.
- **Large** γ (narrow bumps): each point influences only its immediate vicinity $\Rightarrow$ wiggly boundary that can *overfit*, in the limit memorizing the training set.

γ and C interact strongly and must be tuned *together* (a 2-D grid/Bayesian search), since both control the bias–variance balance.



7.3 Kernels for structured data

Intuition

Because a kernel only needs to measure *similarity*, SVMs apply to objects that are not naturally vectors:
- **String kernels** count shared substrings/k-mers — powerful for text and protein/DNA sequences.
- **Graph kernels** compare graphs by shared walks, paths, or subtrees — molecules, social networks.
- **Histogram/χ^2 kernels** compare distributions — the bag-of-visual-words era of computer vision.

This is a genuine strength: design a valid (PSD) similarity for your domain, and the entire SVM machinery applies unchanged. You can also **combine kernels** (sums and products of kernels are kernels) to fuse multiple data sources.

7.4 Scaling kernels to large data

Key idea

Kernel SVMs need the $n \times n$ Gram matrix, which is $O(n^2)$ memory and roughly $O(n^2$–$n^3)$ time — impractical beyond $\sim 10^5$ points. The standard escape is to **approximate the feature map explicitly** and then run a fast *linear* SVM:
- **Random Fourier Features** approximate the RBF kernel with a random low-dimensional map $\tilde{\phi}(\mathbf{x})$ such that $\tilde{\phi}(\mathbf{x})^\top \tilde{\phi}(\mathbf{x}') \approx k(\mathbf{x}, \mathbf{x}')$.
- **Nyström approximation** builds a low-rank Gram approximation from a sampled subset of points.

Both turn a kernel SVM into a scalable linear one with almost the same accuracy.

Common pitfall

Kernel-selection mistakes are the most common SVM failures. (1) **RBF without scaling**: distances are dominated by large-range features — always standardize. (2) **Default** γ **blindly**: scikit-learn's `gamma="scale"` is a sane start but rarely optimal; tune it. (3) **High-degree polynomials**: numerically unstable and prone to overfitting — prefer RBF.

(4) **Kernel on millions of rows**: use linear SVM or random features instead. (5) **Invalid kernels**: a hand-rolled similarity that is not PSD can silently break the optimization.

How this powers ML / AI

Random Fourier features — approximating a kernel by a random nonlinear map — is a conceptual bridge to deep learning: a single wide random layer followed by a linear classifier *is* an approximate kernel machine, foreshadowing the Neural Tangent Kernel view that wide networks behave like kernel methods (Chapter 13). Structured kernels (string, graph) were the dominant approach to sequences and molecules *before* RNNs and graph neural networks, and graph kernels remain competitive baselines. The lesson — "encode domain structure into a similarity" — directly informs how modern architectures bake in inductive biases.

Try it in NumPy

```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True)
pipe = make_pipeline(StandardScaler(), SVC(kernel="rbf"))
grid = {"svc__C": [0.1, 1, 10, 100], "svc__gamma": [1e-3, 1e-2, 1e-1,
    1]}
gs = GridSearchCV(pipe, grid, cv=5).fit(X, y)     # tune C and gamma
   jointly
print("best params:", gs.best_params_, " cv acc:", round(gs.
   best_score_, 3))
```

Chapter summary

- Standard kernels: **linear** (text/high-dim), **polynomial** (interactions), **RBF** (default nonlinear), sigmoid (risky).
- For RBF, γ is inverse bandwidth: small γ underfits (smooth), large γ overfits (wiggly); tune with C.
- Kernels extend SVMs to **strings, graphs, histograms** — any domain with a valid PSD similarity; kernels can be combined.
- Kernel SVMs are $O(n^2)$; scale via **random Fourier features** or **Nyström** + linear SVM.
- Always standardize, and tune the RBF γ jointly with C; random-feature maps prefigure the kernel view of wide networks.

8

Support Vector Regression

Level: Advanced

The maximum-margin idea is not limited to classification. **Support Vector Regression (SVR)** flips the logic: instead of keeping points *outside* a margin, it tries to keep them *inside* a tube around the prediction, ignoring small errors entirely. The result is a flexible, kernelizable, sparse regressor with a distinctive robustness to noise.

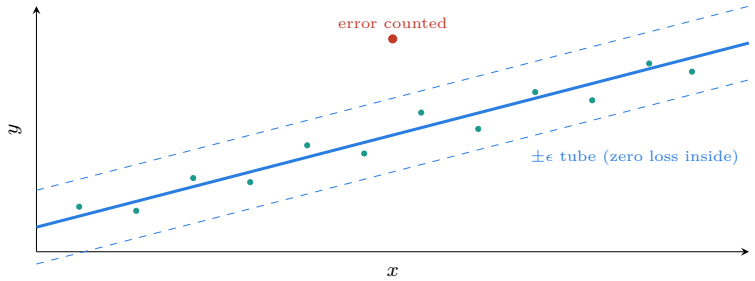
8.1 The ϵ-insensitive tube

Key idea

SVR fits a function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ (or its kernelized version) and penalizes only errors larger than a tolerance ϵ. Inside the **ϵ-tube** ($|y - f(\mathbf{x})| \le \epsilon$) the loss is *zero*; outside, it grows linearly. The objective mirrors the classification SVM:

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

subject to the residuals lying within ϵ plus slacks $\xi_i, \xi_i^* \ge 0$ above/below the tube.



Definition 8.1. The **ϵ-insensitive loss** is $\ell_\epsilon(y, f) = \max(0,\ |y - f| - \epsilon)$. It is the regression analog of the hinge loss: flat (zero) in a band, linear beyond it.

8.2 Support vectors in regression

Intuition

As with classification, the KKT conditions make SVR **sparse**: points strictly *inside* the tube have $\alpha_i = 0$ and are ignored; only points on or outside the tube boundary become support vectors and define f. So ϵ controls sparsity and tube width, while C controls how

hard tube violations are penalized. A wider ϵ means fewer support vectors and a smoother fit; a larger C tracks the data more tightly.

8.3 Kernelized SVR and ν-SVR

Key idea

Everything kernelizes: replace inner products with $k(\mathbf{x}_i, \mathbf{x}_j)$ and SVR fits smooth nonlinear curves, $f(\mathbf{x}) = \sum_{i \in \mathrm{SV}} (\alpha_i - \alpha_i^*)\, k(\mathbf{x}_i, \mathbf{x}) + b$. A useful variant, **$\nu$-SVR**, replaces the unintuitive ϵ with a parameter $\nu \in (0, 1]$ that directly upper-bounds the fraction of points allowed outside the tube and lower-bounds the fraction of support vectors — often easier to set than ϵ.

8.4 Why the tube matters

Intuition

The flat-bottomed loss gives SVR a robustness that squared-error regression lacks: small residuals cost *nothing*, so the fit is not pulled around by every minor wiggle, and the linear (not quadratic) growth outside the tube limits the influence of large outliers. The trade-off is three coupled hyperparameters (C, ϵ or ν, and kernel parameters) and the same $O(n^2)$ scaling as kernel classification.

Common pitfall

SVR pitfalls echo the classifier's, plus one of its own: **scale the target too.** Because ϵ is in the units of y, an ϵ chosen for targets in the thousands is meaningless for targets in fractions — standardize features *and* consider scaling y. Other traps: setting ϵ too large (the model predicts a near-constant), too small (every point becomes a support vector, slow and overfit), and forgetting that SVR, like all kernel methods, struggles past $\sim 10^5$ samples.

How this powers ML / AI

The ϵ-insensitive loss is part of a family of **robust regression losses** that pervade modern ML — the Huber loss in robust statistics and the smooth-L_1 loss used for *bounding-box regression* in object detectors (Fast R-CNN, SSD, YOLO) share its "flat near zero, linear in the tail" shape. Kernel ridge regression and Gaussian process regression are close cousins of SVR in the kernel-methods family, and remain go-to tools for small-data regression with uncertainty. The principle "don't penalize errors you don't care about" recurs whenever we design losses tolerant to noise.

Try it in NumPy

```
import numpy as np
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
rng = np.random.default_rng(0)
```

```
X = np.sort(rng.uniform(0, 10, 200)).reshape(-1, 1)
y = np.sin(X).ravel() + rng.normal(0, 0.1, 200)
model = make_pipeline(StandardScaler(), SVR(kernel="rbf", C=10, gamma
    =0.5, epsilon=0.1))
model.fit(X, y)
print("R^2:", round(model.score(X, y), 3))
print("# support vectors:", model[-1].support_.size, "of", len(X))
```

Chapter summary

- **SVR** fits a function with an **ϵ-insensitive tube**: errors within $\pm\epsilon$ cost nothing, beyond it grow linearly.
- The objective $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum(\xi_i + \xi_i^*)$ mirrors classification; ϵ sets tube width/sparsity, C sets the penalty.
- Only points on/outside the tube are **support vectors**; SVR is sparse and **kernelizable** for nonlinear fits.
- **ν-SVR** swaps ϵ for ν, bounding the fraction of outliers/support vectors directly.
- Scale features *and* target; the ϵ-insensitive loss is kin to the smooth-L_1 losses used in modern detectors.

9 Solving SVMs: Optimization Algorithms

Level: Expert

An SVM is only as useful as our ability to solve its optimization problem efficiently. The dual is a quadratic program with n variables and a dense $n \times n$ matrix — naively intractable for large data. This chapter explains the algorithms that make SVMs practical: general QP solvers, the elegant SMO decomposition that powers LIBSVM, and the linear-time methods (LIBLINEAR, Pegasos) for massive linear problems.

9.1 The optimization landscape

Key idea

SVM training is a **convex** problem, so any solver reaches the *global* optimum — no local minima, no random restarts. The practical question is *which* solver, and that depends on the regime:

- **Kernel SVM, moderate** n: solve the dual QP — decomposition methods (SMO).
- **Linear SVM, large** n: solve the primal — coordinate descent (LIBLINEAR) or stochastic subgradient (Pegasos).

9.2 Quadratic programming and its limits

Intuition

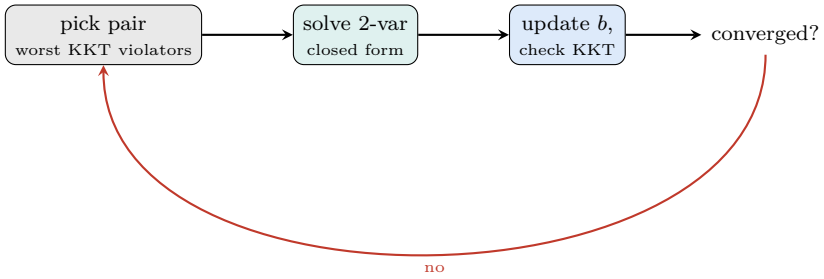
The dual is a textbook quadratic program (QP): maximize a concave quadratic in α subject to one linear equality and box constraints $0 \le \alpha_i \le C$. Off-the-shelf QP solvers work for small problems, but they need the full Gram matrix in memory ($O(n^2)$) and run in roughly $O(n^3)$ time — hopeless beyond a few thousand points. We need a method that never materializes the whole matrix.

9.3 Sequential Minimal Optimization (SMO)

Key idea

SMO (Platt, 1998) is the breakthrough that made kernel SVMs practical. It exploits the equality constraint $\sum_i \alpha_i y_i = 0$: because of it, you cannot change one α alone, but you *can* change the smallest meaningful set — a **pair** (α_i, α_j). For a pair, the subproblem has a **closed-form analytic solution** (no inner solver needed). SMO repeatedly: (1) picks a

pair that most violates the KKT conditions, (2) optimizes it exactly, (3) repeats until all KKT conditions hold within tolerance.



Intuition

SMO's genius is that the most expensive step in QP — solving a constrained subproblem — becomes a few lines of arithmetic for a pair. It uses little memory (compute kernel entries on demand, cache hot ones) and scales to tens of thousands of points. **LIBSVM**, the library behind scikit-learn's `SVC`/`SVR`, is essentially a polished SMO with smart working-set selection and caching.

9.4 Large-scale linear SVMs

Key idea

When the kernel is linear, you can keep $\mathbf{w}$ explicitly and avoid the Gram matrix entirely, giving (near) linear-time training:

- **LIBLINEAR** uses dual or primal **coordinate descent**, handling millions of samples/features (ideal for text).
- **Pegasos** performs **stochastic subgradient descent** on the primal, with run time essentially independent of dataset size.
- **SGDClassifier(loss="hinge")** in scikit-learn is the mini-batch SGD route, streaming and online-friendly.

Rule of thumb: $n \gtrsim 10^5$ and you want a linear model $\Rightarrow$ use LIBLINEAR/SGD, not kernel SVC.

Common pitfall

Performance traps. (1) **Using `SVC` (kernel) on huge data**: training time explodes super-linearly — switch to `LinearSVC` or `SGDClassifier`. (2) **Ignoring the cache**: for kernel SVMs, a too-small kernel cache thrashes; raise `cache_size`. (3) **Loose tolerance vs. speed**: tightening `tol` too far wastes time for no accuracy gain. (4) **Not shrinking**: LIBSVM's shrinking heuristic (dropping settled variables) speeds convergence — keep it on. (5) Forgetting that `LinearSVC` and `SVC(kernel="linear")` use *different* solvers and can give slightly different boundaries.

How this powers ML / AI

SMO's decomposition idea — repeatedly solving tiny exactly-solvable subproblems — recurs across optimization in ML, from coordinate descent in Lasso to block updates in large-scale solvers. More importantly, **stochastic subgradient descent** (Pegasos) for

the SVM primal is the same algorithm family that trains every modern neural network: sample a mini-batch, take a noisy gradient step, repeat. Studying Pegasos — with its $1/t$ step size and convergence guarantees on a convex objective — is the cleanest possible introduction to why SGD works, before the non-convex complications of deep learning.

Try it in NumPy

```python
import numpy as np, time
from sklearn.svm import SVC, LinearSVC
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=40000, n_features=50,
    random_state=0)
for name, model in [("kernel SVC(linear)", SVC(kernel="linear")),
                    ("LinearSVC", LinearSVC(max_iter=5000))]:
    t = time.time(); model.fit(X, y)
    print(f"{name:>20}: {time.time()-t:5.1f}s  acc={model.score(X, y)
    :.3f}")
# LinearSVC is dramatically faster at this scale
```

Chapter summary

- SVM training is **convex** (global optimum guaranteed); the choice of solver depends on kernel vs. linear and on n.
- The dual is a **QP** needing the $O(n^2)$ Gram matrix — generic solvers don't scale.
- **SMO** optimizes one **pair** of multipliers at a time in closed form; it powers **LIBSVM** (scikit-learn's `SVC`).
- For large **linear** SVMs, use **LIBLINEAR** (coordinate descent) or **Pegasos**/SGD (stochastic subgradient).
- Match solver to scale; Pegasos/SGD on the convex SVM is the cleanest lens on the SGD that trains neural nets.

PART IV

Practice and Theory

10

Multiclass, Probabilities, and Practical SVMs

Level: Expert

An SVM is natively a *binary*, *non-probabilistic* classifier. Real problems have many classes and often need probabilities. This chapter covers the multiclass schemes, how to extract calibrated probabilities, and the end-to-end craft — scaling, tuning, handling imbalance, and building leak-free pipelines — that turns an SVM from a textbook idea into a dependable model.

10.1 From binary to multiclass

Key idea

SVMs handle $K > 2$ classes by combining binary classifiers:

- **One-vs-Rest (OvR)**: train K classifiers, each "class k vs. all others"; predict the highest-scoring. Cheap (K models) but classes are imbalanced within each problem.
- **One-vs-One (OvO)**: train $\binom{K}{2}$ classifiers, one per pair; predict by majority vote. More models but each is small and balanced — this is what **LIBSVM/`SVC`** uses.

For large K, OvO's $O(K^2)$ count grows, but each model trains on only the two relevant classes, so total cost is often manageable.

10.2 Getting probabilities

Intuition

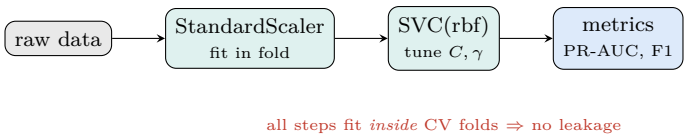
The raw score $\mathbf{w}^\top \mathbf{x} + b$ is a signed distance, not a probability. To get $P(y \mid \mathbf{x})$, SVMs use **Platt scaling**: fit a logistic function $\sigma(a\, f(\mathbf{x}) + b)$ to the SVM scores on held-out data. scikit-learn does this when you pass `probability=True` — but be aware it runs an internal cross-validation (slower) and the resulting probabilities can be inconsistent with the `decision_function` ranking. For ranking/thresholding you often don't need probabilities at all; use `decision_function` directly.

10.3 The practitioner's checklist

Key idea

A reliable SVM workflow:

1. **Standardize features** (zero mean, unit variance) — mandatory, especially for RBF.
2. **Choose a kernel**: linear for high-dim/sparse (text); RBF as the nonlinear default.
3. **Tune** C (and γ for RBF) on a **log grid** by cross-validation; they interact, so search jointly.
4. **Handle imbalance** with `class_weight="balanced"` and judge with the right metric (PR-AUC/F1, not accuracy).
5. **Wrap everything in a `Pipeline`** so scaling is fit inside each CV fold (no leakage).
6. **Mind scale**: kernel `SVC` for up to $\sim 10^4$–10^5 rows; `LinearSVC`/SGD beyond.



10.4 One-class SVM for anomaly detection

Intuition

A **one-class SVM** learns the support of "normal" data: it finds a boundary (in kernel feature space) enclosing most training points, flagging anything outside as an anomaly. With no labels needed, it is a classic tool for novelty and outlier detection (fraud, fault monitoring, intrusion). The parameter ν sets the expected fraction of outliers. It reappears in Chapter 13 as the surprising mathematical model for self-supervised contrastive learning.

Common pitfall

Production gotchas. (1) **Scaling leakage**: fitting the scaler on all data before splitting leaks test statistics — always scale inside the pipeline/folds. (2) **`probability=True` surprises**: it retrains with CV and can disagree with `decision_function`; only enable it if you truly need probabilities. (3) **Imbalance ignored**: a 99/1 problem scores 99% by predicting the majority — set `class_weight` and use PR-AUC. (4) **Default `gamma`** left untuned. (5) **Kernel SVM on too-large data**, where training silently takes hours — subsample or go linear.

How this powers ML / AI

The practical scaffolding here — standardize, pipeline, grid/Bayesian search, calibrate, threshold, judge with the right metric — is identical to MLOps for any model, deep learning included, and uses the same tooling (scikit-learn pipelines, Optuna). Platt scaling, invented to calibrate SVMs, is now applied to *neural networks* too (alongside temperature scaling) to fix their overconfidence. And the one-class SVM is not just a legacy tool: it is the conceptual template for modern self-supervised representation learning (Chapter 13).

Try it in NumPy

```python
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_wine
X, y = load_wine(return_X_y=True)                    # 3 classes
pipe = make_pipeline(StandardScaler(),
                     SVC(kernel="rbf", class_weight="balanced",
                         decision_function_shape="ovo"))
grid = {"svc__C": [0.1, 1, 10], "svc__gamma": ["scale", 0.01, 0.1]}
gs = GridSearchCV(pipe, grid, cv=5).fit(X, y)
print("best:", gs.best_params_, " cv acc:", round(gs.best_score_, 3))
```

Chapter summary

- Multiclass via **One-vs-Rest** (K models) or **One-vs-One** ($\binom{K}{2}$ models, used by `SVC`).
- SVMs aren't probabilistic; get $P(y \mid \mathbf{x})$ via **Platt scaling** (`probability=True`) — or use `decision_function` for ranking.
- Always **standardize**, **pipeline** to avoid leakage, tune C, γ jointly, and handle imbalance with `class_weight`.
- **One-class SVM** (ν) detects anomalies without labels — and models contrastive learning later.
- Use kernel `SVC` up to $\sim 10^5$ rows; switch to `LinearSVC`/SGD beyond.

11

Margins, VC Dimension, and Generalization

Level: Expert

Why does maximizing the margin actually help on *unseen* data? The answer is one of the crown jewels of machine learning: **statistical learning theory**, developed largely by Vapnik alongside the SVM. This chapter explains VC dimension, the structural-risk-minimization principle, and the margin-based bounds that justify the SVM — the theory that told us, decades before deep learning, why some models generalize.

11.1 The generalization question

Definition 11.1. A learner's **generalization error** is its expected error on new data drawn from the same distribution. The gap between *training* error and *generalization* error is what we must control: a model that fits training data perfectly but generalizes poorly has **overfit**.

11.2 VC dimension and capacity

Key idea

The **Vapnik–Chervonenkis (VC) dimension** measures a model class's **capacity** — the largest number of points it can *shatter* (classify correctly under every possible labeling). Higher VC dimension means more flexibility but greater risk of overfitting. VC theory bounds the generalization gap roughly as

$$\text{test error} \;\lesssim\; \text{train error} \;+\; O\!\left(\sqrt{\frac{h\,\log n}{n}}\right),$$

where h is the VC dimension and n the sample size. Control h relative to n and you control overfitting.

Intuition

A line in the plane has VC dimension 3: it can shatter any 3 points (in general position) but not 4. The danger is that flexible models (high h) can fit noise. The classical recipe is **Structural Risk Minimization (SRM)**: among model classes of increasing capacity, choose the one that best balances training fit against capacity — exactly the bias–variance tradeoff, made rigorous.

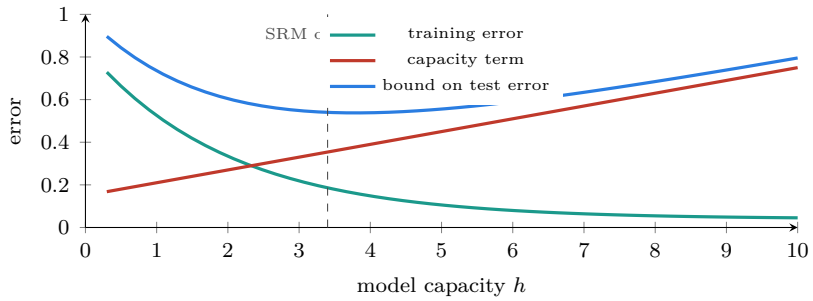
11.3 Why large margins generalize

Key idea

Here is the SVM's theoretical punchline. For **large-margin** separators, the relevant capacity is *not* the dimension of the space but the inverse squared margin. A **margin bound** states (informally)

$$\text{test error} \lesssim \frac{R^2/\rho^2}{n},$$

where ρ is the margin and R bounds the data radius. The remarkable consequence: a classifier with a **big margin** generalizes well *even in very high- (or infinite-) dimensional* feature spaces. This is precisely why the kernel trick does not doom us to overfitting — the margin, not the dimension, governs generalization.



11.4 The representer theorem

Intuition

A second pillar from theory: the **representer theorem** guarantees that the function minimizing a regularized loss over an RKHS has the finite form $f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x})$ — a weighted sum of kernels centered at the *training points*. Even when the feature space is infinite-dimensional, the solution lives in the n-dimensional span of the data. This is what makes kernel methods *computable* and explains, mathematically, why support vectors suffice.

Common pitfall

Theory caveats. (1) VC bounds are often **loose** — they explain *why* margins help directionally but rarely give tight numbers; trust cross-validation for actual estimates. (2) The bounds assume **i.i.d. data** from a fixed distribution; under distribution shift they don't apply. (3) **High-dimensional intuition fails**: "more features must overfit" is wrong for large-margin classifiers — the whole point of the margin bound. (4) Classical VC theory famously **under-predicts** deep learning's generalization, which spurred new tools (margin distributions, NTK, double descent) — a frontier the next chapters touch.

How this powers ML / AI

SVM theory *is* the foundation of modern generalization research. The large-margin principle directly inspired margin analyses of neural networks; the inability of classical VC theory to explain why over-parameterized deep nets generalize launched the study of implicit regularization, margin distributions, and **double descent**. And the cleanest bridge

— the Neural Tangent Kernel — recasts wide networks as kernel machines, importing the SVM's margin and RKHS guarantees into deep learning (Chapter 13). When researchers prove "non-vacuous generalization bounds" for neural networks today, they often do it by relating the network to its equivalent kernel SVM.

Try it in NumPy

```python
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=600, n_features=20,
    n_informative=6, random_state=0)
clf = SVC(kernel="linear", C=1.0).fit(X, y)
w = clf.coef_[0]; margin = 1/np.linalg.norm(w)
R = np.linalg.norm(X, axis=1).max()
print("margin rho =", round(margin, 3), " radius R =", round(R, 2))
print("capacity proxy R^2/rho^2 =", round((R**2)/(margin**2), 1))
print("cv accuracy =", round(cross_val_score(clf, X, y, cv=5).mean(),
    3))
```

Chapter summary

- **Generalization** depends on controlling model **capacity**; the **VC dimension** quantifies capacity (shattering).
- **Structural Risk Minimization** balances training fit against capacity — the rigorous bias–variance tradeoff.
- **Margin bounds** show error scales like $R^2/(\rho^2 n)$: a **large margin** generalizes even in infinite dimensions.
- The **representer theorem** keeps RKHS solutions finite ($\sum_i \alpha_i k(\mathbf{x}_i, \cdot)$), making kernels computable.
- Classical theory underexplains deep nets, motivating NTK/margin/double-descent research that links back to SVMs.

PART V

Frontier and Applications

12

SVMs versus the World: When to Use What

Level: Expert

When should you reach for an SVM today? This chapter places SVMs honestly in the modern landscape — against logistic regression, tree ensembles, and deep networks — with a clear-eyed account of their strengths, weaknesses, and the niches where they remain the best choice. The short version: SVMs shine on small-to-medium, high-dimensional, clean problems, and fade where data is huge or raw and perceptual.

12.1 Strengths and weaknesses

Strengths	Weaknesses
Effective in high dimensions ($p \gg n$)	Scales poorly with n ($O(n^2)$ kernels)
Strong generalization (margin theory)	Sensitive to C, γ and feature scaling
Kernel trick $\Rightarrow$ flexible nonlinearity	No native probabilities (needs calibration)
Convex: unique global optimum	Hard to interpret with nonlinear kernels
Sparse (support vectors only)	Outliers/noise can dominate; slow to tune

12.2 SVM vs logistic regression

Intuition

They are close cousins — both linear, both regularized, differing mainly in loss (hinge vs. log). Use **logistic regression** when you need calibrated probabilities, interpretable coefficients, or streaming/online updates. Use a **linear SVM** when you want maximum-margin robustness and only decisions, especially in very high dimensions like text. In practice their accuracies are often similar; the choice hinges on probabilities and interpretability versus margin robustness.

12.3 SVM vs tree ensembles

Key idea

On **tabular** data with mixed types, missing values, and irregular relationships, **gradient-boosted trees** (XGBoost/LightGBM) usually win: they need no scaling, handle categoricals and missingness natively, capture interactions automatically, and scale to large n. SVMs compete when features are **dense, continuous, and high-dimensional** (e.g. after a good numeric featurization), when the dataset is **small-to-medium**, and when a **well-chosen kernel** encodes real structure. For a generic new table, try boosting first; reach for an RBF-SVM as a strong alternative baseline.

12.4 SVM vs deep learning

Intuition

On **raw perceptual data** — images, audio, text, sequences — deep networks decisively beat SVMs: they *learn* representations end-to-end, while SVMs need a fixed (kernel or hand-crafted) feature space and choke on the data volume. But for **small datasets**, **high-dimensional structured features**, or settings demanding a **convex, reproducible** model with theory-backed guarantees, SVMs remain excellent — and far cheaper to train. A classic hybrid still used today: take features from a pretrained deep network and classify them with an SVM (Chapter 13).

12.5 A decision guide

Situation	Reach for
Small/medium, high-dim, dense features	SVM (RBF or linear)
Text classification, sparse high-dim	Linear SVM (or logistic regression)
Need calibrated probabilities / coefficients	Logistic regression
Generic tabular, mixed types, large n	Gradient-boosted trees
Images / audio / text (raw), big data	Deep neural networks
Few labels on top of deep features	Deep features + SVM
Unlabeled novelty / outlier detection	One-class SVM (or Isolation Forest)

Common pitfall

Don't misuse SVMs by reflex. Putting a kernel `SVC` on a million rows (it crawls), feeding raw pixels to an RBF-SVM (no learned features — a CNN dominates), skipping standardization (RBF distances meaningless), or shipping an uncalibrated SVM where probabilities are needed — these are the recurring failures. Equally, don't *dismiss* SVMs: on a clean

2,000-row, 500-feature biology dataset, a tuned RBF-SVM can beat a fussy neural net at a fraction of the effort, with theory guaranteeing why.

How this powers ML / AI

The right framing is **inductive bias**, the theme of modern ML. SVMs encode "large-margin separation in a fixed kernel-defined geometry" — ideal when that geometry matches your data and the dataset is modest. Trees encode "axis-aligned, scale-free partitions" for tables; CNNs/transformers encode locality and attention for perception. Knowing each model's bias — and that an infinitely wide network is itself a kernel machine — is what lets an expert pick, combine, and reason about models rather than chase fashions. The capstone shows how the SVM's biases live on inside deep learning.

Try it in NumPy

```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True)
models = {"linear SVM": make_pipeline(StandardScaler(), SVC(kernel="
    linear")),
          "RBF SVM":    make_pipeline(StandardScaler(), SVC(kernel="
    rbf", gamma="scale")),
          "logistic":   make_pipeline(StandardScaler(),
    LogisticRegression(max_iter=500)),
          "hist-GBM":   HistGradientBoostingClassifier()}
for name, m in models.items():
    print(f"{name:>11}: {cross_val_score(m, X, y, cv=5).mean():.3f}")
```

Chapter summary

- SVMs excel in **high dimensions** and **small/medium** clean data; they scale poorly with n and need careful scaling/tuning.
- Vs **logistic regression**: similar accuracy; choose logistic for probabilities/interpretability, SVM for margin robustness.
- Vs **trees**: boosting usually wins on generic tabular data; SVMs compete on dense high-dim features.
- Vs **deep learning**: nets win on raw perceptual big data; SVMs win on small/structured problems and as a head on deep features.
- Choose by **inductive bias** and data size; use the decision guide, and don't reflexively use or dismiss SVMs.

13

Support Vector Machines in Machine Learning, Deep Learning, and AI

Level: Capstone

This capstone gathers the threads. It is tempting to file SVMs under "what we used before deep learning," but that misses the deeper story: the ideas SVMs crystallized — **maximum margin**, **kernels**, **convex duality**, **generalization theory** — are woven through modern machine learning, and recent results show that deep networks themselves, in the right limit, *are* support vector machines. We tour the SVM's living legacy, from hinge losses in neural nets to the Neural Tangent Kernel, contrastive learning, and robustness certificates.

13.1 Where SVMs are still the right tool

Key idea

SVMs remain a first-class choice for **small-to-medium, high-dimensional, structured** problems: text and document classification, bioinformatics (gene expression, protein/DNA with string kernels), medical and scientific datasets with few but rich samples, and **anomaly detection** (one-class SVM). They are the model of choice when you need a *convex, reproducible* classifier with theory-backed guarantees and limited data. They are also a standard **lightweight head** on top of features from large pretrained models.

13.2 The hinge loss and large-margin deep learning

Intuition

The SVM's hinge loss did not retire — it moved into deep learning. Replacing a network's softmax/cross-entropy head with a **linear SVM (L2-hinge) head** ("Deep Learning using Linear Support Vector Machines," Tang 2013) sometimes improves classification. More broadly, the **large-margin principle** powers metric learning: **triplet** and **contrastive** losses pull matching pairs together and push mismatched pairs apart *by a margin*, and margin-based softmax variants (**large-margin softmax, ArcFace, CosFace**) drive state-of-the-art face recognition. "Be correct by a margin, or pay" is everywhere.

13.3 Kernels and deep features

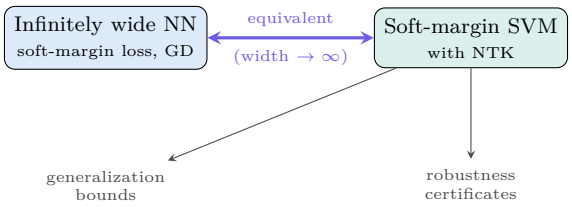
Intuition

The classic and still-useful hybrid: extract a representation from a deep network (the penultimate layer), then train an SVM on those features. This combines deep *representation learning* with the SVM's strong *small-data* classification and convex optimization — handy for transfer learning when labels are scarce. The wider lesson is that kernels and deep nets are complementary: nets learn the feature map ϕ from data; kernels specify it by hand. Each is powerful where the other is weak.

13.4 The deep equivalence: NTK = SVM

Key idea

The most striking modern connection: in the **infinite-width limit**, a neural network trained by gradient descent behaves like a **kernel machine** with the **Neural Tangent Kernel (NTK)**. Recent theory (Chen et al., 2021) sharpens this for classification: an infinitely wide network trained with **soft-margin (hinge) loss** by subgradient descent is *equivalent to a soft-margin SVM with the NTK*. Every finite-width network trained with such losses is *approximately* a kernel machine. The SVM is thus not a predecessor of deep learning but, in a precise sense, its infinite-width skeleton.



13.5 Contrastive learning as one-class SVMs

Intuition

Self-supervised **contrastive learning** — the engine behind modern representation learning (SimCLR, MoCo, and the embeddings inside multimodal models) — has a surprising SVM interpretation. Recent work shows that a model trained with InfoNCE-style contrastive losses approximately implements an **ensemble of one-class SVMs** in NTK feature spaces: each gradient update behaves like a one-class SVM's primal solution, and the dual **Lagrange multipliers** measure the importance of hard positive/negative triplets. The SVM's machinery — support vectors as the "hard" examples, multipliers as importance weights — reappears at the heart of how today's representations are learned.

13.6 Theory: bounds and certified robustness

Intuition

The SVM–NTK equivalence is not just elegant — it is *useful*. Because kernel machines come with margin-based generalization bounds and exact robustness analysis, researchers transfer these guarantees to neural networks via their equivalent kernel SVMs: **non-vacuous**

generalization bounds, **robustness certificates** against input perturbations, and certified defenses against **label poisoning** all use the network-as-kernel-SVM view. The SVM's mature theory is being borrowed to make deep learning safer and better understood.

13.7 The SVM's place in the AI stack

Domain	Best-fit model	Role of SVM ideas
Small/medium structured data	SVM (RBF/linear)	primary model
Text / bioinformatics	linear / string-kernel SVM	primary or strong baseline
Anomaly / novelty detection	one-class SVM	primary unsupervised model
Face recognition / retrieval	deep nets	margin losses (ArcFace)
Representation learning	contrastive deep nets	one-class-SVM dynamics (NTK)
DL theory & safety	wide networks	NTK = kernel SVM; certificates

Common pitfall

Two opposite errors. **SVM nostalgia:** forcing kernel SVMs onto huge raw-perceptual datasets where deep nets and their learned features dominate — the wrong tool and the wrong scale. **SVM dismissal:** assuming SVMs are obsolete — they win on small high-dimensional data, anchor anomaly detection, and, through the NTK and margin-loss connections, underpin how we *understand* and *secure* deep learning. The expert keeps the SVM both as a practical model and as a theoretical lens.

How this powers ML / AI

The big picture. SVMs bequeathed four enduring ideas to AI: (1) the **maximum-margin principle**, now driving contrastive and face-recognition losses; (2) the **kernel viewpoint**, which matured into Gaussian processes and resurfaced as the NTK describing wide neural networks; (3) **convex duality and KKT**, the optimization grammar behind sparsity, robustness, and constrained learning; and (4) **margin-based generalization theory**, still the template for explaining and certifying modern models. Far from being superseded, the Support Vector Machine turned out to be a skeleton key: master its margin, its kernels, and its duality, and you hold a unifying lens on machine learning, deep learning, and AI.

Chapter summary

- SVMs remain primary for **small/medium high-dim** data, **text/bioinformatics**, and **anomaly detection**.
- The **hinge/large-margin** principle lives in deep learning: SVM heads, triplet/contrastive losses, ArcFace/CosFace.
- Classic hybrid: **deep features + SVM** for small-label transfer learning.

- **NTK = SVM**: infinitely wide nets trained with soft-margin loss equal kernel SVMs — importing bounds and **robustness certificates**.
- **Contrastive learning** approximates ensembles of **one-class SVMs** (NTK); the SVM's legacy is margin, kernels, duality, and theory.

A

Formulas, Kernels, and Hyperparameter Reference

A compact reference for the formulas, kernels, and knobs used throughout the book.

A.1 The SVM at a glance

Object	Expression
Decision rule	$\hat{y} = \operatorname{sgn}(\mathbf{w}^\top \mathbf{x} + b)$
Margin width	$2/\|\mathbf{w}\|$
Hard-margin primal	$\min \frac{1}{2}\|\mathbf{w}\|^2$ s.t. $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$
Soft-margin primal	$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$ s.t. $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0$
Hinge-loss form	$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$
Dual	$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j),\ \ 0 \leq \alpha_i \leq C,$ $\sum_i \alpha_i y_i = 0$
Weight from dual	$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
Kernel prediction	$\hat{y}(\mathbf{x}) = \operatorname{sgn}\big(\sum_{i \in \mathrm{SV}} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\big)$
KKT (compl. slack.)	$\alpha_i[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i] = 0$

A.2 Kernel reference

Kernel	$k(\mathbf{x}, \mathbf{x}')$	Notes
Linear	$\mathbf{x}^\top \mathbf{x}'$	text, $p \gg n$; fast, interpretable
Polynomial	$(\gamma\, \mathbf{x}^\top \mathbf{x}' + r)^d$	degree-d interactions; tune d, γ, r
RBF / Gaussian	$\exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$	default nonlinear; tune γ, C
Sigmoid	$\tanh(\gamma\, \mathbf{x}^\top \mathbf{x}' + r)$	not always PSD; use with care
String / graph	shared substrings / walks	structured (sequences, molecules)

A.3 Hyperparameter cheat-sheet

Parameter	Effect	Typical range
`C`	inverse regularization; large = fit hard, narrow margin	10^{-2}–10^{3} (log grid)
`gamma` (RBF)	inverse bandwidth; large = wiggly, overfits	10^{-4}–10^{1}, or `"scale"`
`degree` (poly)	polynomial degree	2–4
`epsilon` (SVR)	tube half-width (units of y)	set per target scale
`nu` (ν-SVM)	bounds fraction of SV/outliers	$(0, 1]$
`class_weight`	reweight classes for imbalance	`"balanced"`
`kernel`	similarity function	linear / rbf / poly

A.4 Decision: which solver / model

Regime	Use
Kernel SVM, $n \lesssim 10^5$	`SVC`/`SVR` (LIBSVM, SMO)
Linear SVM, large n	`LinearSVC` (LIBLINEAR) or `SGDClassifier(loss="hinge")`
Approximate kernel, large n	random Fourier features / Nyström + linear SVM
Probabilities needed	`probability=True` (Platt) or calibrate separately
Anomaly detection	`OneClassSVM` (ν)

A.5 Key references

- Boser, Guyon, Vapnik, "A Training Algorithm for Optimal Margin Classifiers" (1992).
- Cortes & Vapnik, "Support-Vector Networks," *Machine Learning* (1995).
- Vapnik, *The Nature of Statistical Learning Theory* (1995); *Statistical Learning Theory* (1998).
- Platt, "Sequential Minimal Optimization" (1998); "Probabilistic Outputs for SVMs" (1999).
- Schölkopf & Smola, *Learning with Kernels* (2002); Shawe-Taylor & Cristianini, *Kernel Methods* (2004).
- Schölkopf et al., "Estimating the Support of a High-Dimensional Distribution" (one-class SVM, 2001).
- Shalev-Shwartz et al., "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM" (2007).
- Jacot et al., "Neural Tangent Kernel" (2018); Chen et al., "On the Equivalence between Neural Network and Support Vector Machine" (2021).
- Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*; James et al., *An Introduction to Statistical Learning*.