

Support Vector Machines with Kernels

Lecture Notes Consolidated with Claude AI — CS6140 Machine Learning

Based on Andrew Ng’s CS229 lecture notes (Stanford; Part V, *Support Vector Machines*, and the companion *Simplified SMO* notes), Bingyu Wang’s SVM notes for this course, and Platt’s SMO paper (1998).
Notation (as in previous notes): a datapoint $\mathbf{x}$ is a $1 \times D$ row vector, $\mathbf{w}$ is $D \times 1$, labels are $y \in \{-1, +1\}$ (as for the perceptron), and N is the number of training points.

1 What Is an SVM? Margins and Support Vectors

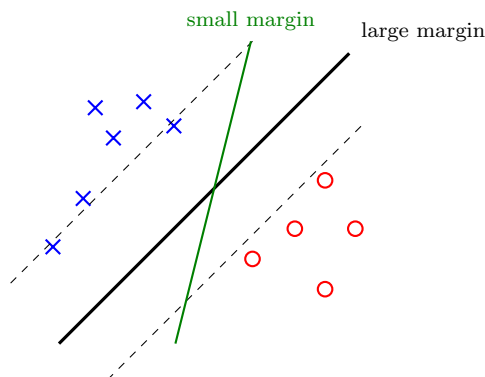
A support vector machine is a **linear classifier**

$$h(\mathbf{x}) = \text{sign}(\mathbf{xw} + b), \quad f(\mathbf{x}) = \mathbf{xw} + b \quad (\text{the score}), \quad (1)$$

like logistic regression and the perceptron. When the data is linearly separable, many hyperplanes separate it; the SVM’s question is *where exactly* to put the hyperplane. Its answer uses the **margin**: the distance from the hyperplane to the *closest* training points.

1.1 The Margin Decides Where the Hyperplane Goes

- The score $f(\mathbf{x})$ measures *confidence*: a point with $f(\mathbf{x}) \gg 0$ is confidently positive, one with $f(\mathbf{x}) \approx 0$ sits on the fence. A hyperplane that passes close to some training points is fragile: a slightly different test point near them may land on the wrong side.
- So the SVM pushes the hyperplane *as far as possible from the closest points of both classes*: it maximizes the margin. Picture the widest empty band (“street”) that fits between the two classes; the hyperplane runs down its middle.
- Only the closest points matter for this. The width and position of the band are fixed by the points on its edges; the points farther away play no role.



Both lines separate the data. The black one leaves a wide empty band (dashed) between the classes; the green one passes close to several points.

1.2 Functional and Geometric Margin

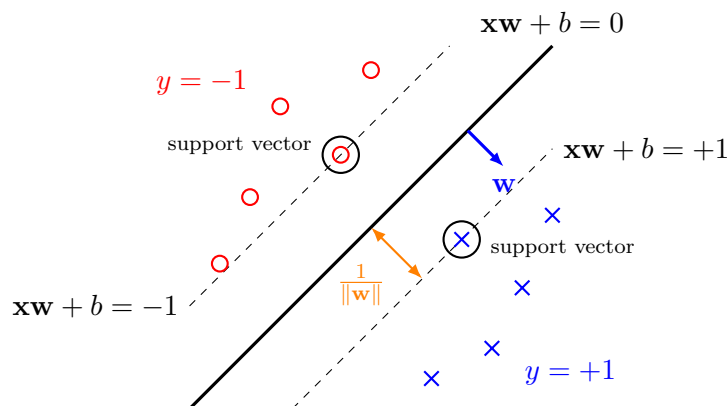
For a training point $(\mathbf{x}_i, y_i)$:

- **Functional margin:** $\hat{\gamma}_i = y_i(\mathbf{x}_i \mathbf{w} + b)$. It is positive exactly when the point is classified correctly, and large when it is classified confidently. But it is not a distance: replacing $(\mathbf{w}, b)$ by $(2\mathbf{w}, 2b)$ doubles every functional margin without moving the hyperplane at all.
- **Geometric margin:** the actual (signed) distance from $\mathbf{x}_i$ to the hyperplane $\{\mathbf{x} : \mathbf{x}\mathbf{w} + b = 0\}$. As in the Perceptron note, $\mathbf{w}$ is the normal vector of the hyperplane, and the signed distance is the score divided by $\|\mathbf{w}\|$:

$$\gamma_i = \frac{y_i(\mathbf{x}_i \mathbf{w} + b)}{\|\mathbf{w}\|} = \frac{\hat{\gamma}_i}{\|\mathbf{w}\|}, \quad (2)$$

unchanged by rescaling $(\mathbf{w}, b)$.

- The **margin of the hyperplane** is that of the closest point: $\gamma = \min_i \gamma_i$. Maximizing γ over all hyperplanes is the SVM.



The maximum-margin hyperplane (solid) and the two margin hyperplanes (dashed), each at distance $1/\|\mathbf{w}\|$. The points on the dashed lines are the **support vectors**; the band between the dashed lines contains no training points.

Key point: Support vectors. The maximum-margin hyperplane is determined entirely by the training points closest to it — the **support vectors**, on the edges of the band. They “support” the separating hyperplane: moving one of them generally moves the hyperplane, while deleting any other point (or moving it, as long as it stays outside the margin) changes nothing. The SVM algorithm specifically finds these points, and the classifier is built only from them. Often they are a relatively small subset of the training data.

Two more ideas make the SVM one of the most successful classifiers of the 1990s–2000s:

- **Soft margin** (§3): when the data is not separable, allow a few points to violate the margin, at a cost (Cortes & Vapnik 1995).
- **Kernels** (§4): the optimization is solved in its *dual* form, which touches the data only through dot products $\mathbf{x}_i \mathbf{x}_j^T$. Replacing them with a kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ (Kernels note) gives a maximum-margin hyperplane in the kernel’s feature space — a nonlinear classifier in the original features, trained by the same code.

2 The Optimal Margin Classifier and Its Dual

2.1 The Primal Problem

Assume for now the data is linearly separable. We want the hyperplane with the largest geometric margin:

$$\max_{\mathbf{w}, b} \gamma \quad \text{s.t.} \quad \frac{y_i(\mathbf{x}_i \mathbf{w} + b)}{\|\mathbf{w}\|} \geq \gamma, \quad i = 1, \dots, N. \quad (3)$$

Fix the scale. Since $(\mathbf{w}, b)$ and $(c\mathbf{w}, cb)$ describe the same hyperplane, we may choose the scale so that the closest points have functional margin exactly 1: $\min_i y_i(\mathbf{x}_i \mathbf{w} + b) = 1$. Then the geometric margin is $\gamma = 1/\|\mathbf{w}\|$, and maximizing it is the same as minimizing $\|\mathbf{w}\|$, or (more conveniently) $\frac{1}{2}\|\mathbf{w}\|^2$:

$$\min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{x}_i \mathbf{w} + b) \geq 1, \quad i = 1, \dots, N \quad (4)$$

Key point: The margin is the distance from the boundary to the closest points, $1/\|\mathbf{w}\|$, and the empty band is $2/\|\mathbf{w}\|$ wide. Every training point must be outside the band, on its own side: $y_i(\mathbf{x}_i \mathbf{w} + b) \geq 1$. Among all such hyperplanes, the SVM picks the one with the smallest $\|\mathbf{w}\|$, i.e. the widest band.

Why $1/\|\mathbf{w}\|$, directly. Take a support vector $\mathbf{x}_+$ on the line $\mathbf{x}\mathbf{w} + b = +1$, and let $\mathbf{x}_-$ be the point on $\mathbf{x}\mathbf{w} + b = -1$ straight across the band from it (the dashed lines in the figure of §1.2). Subtracting the two equations, $(\mathbf{x}_+ - \mathbf{x}_-)\mathbf{w} = 2$. The step from $\mathbf{x}_-$ to $\mathbf{x}_+$ crosses the band along the normal $\mathbf{w}$, so $\mathbf{x}_+ - \mathbf{x}_- = 2\rho \mathbf{w}^T / \|\mathbf{w}\|$, where ρ is the margin (half the band width). Then $2\rho \mathbf{w}^T \mathbf{w} / \|\mathbf{w}\| = 2\rho \|\mathbf{w}\| = 2$, so $\rho = 1/\|\mathbf{w}\|$.

Equation 4 is the SVM **primal** problem. The objective is a convex quadratic and the constraints are linear, so it has a single global optimum, and a generic quadratic programming (QP) solver could solve it directly. We go to the **dual** instead, for two reasons: (1) the dual needs only dot products between datapoints, which brings in kernels (§4); (2) the dual has a structure that a simple, fast algorithm (SMO, §5) can exploit. Getting there takes three steps: write the Lagrangian, minimize it over $\mathbf{w}, b$, and substitute back.

2.2 The Lagrangian

In the PCA note we handled one equality constraint ($\mathbf{e}^T \mathbf{e} = 1$) with one Lagrange multiplier. Here there are N *inequality* constraints; each gets a multiplier α_i , which must be **nonnegative**. Writing each constraint as $1 - y_i(\mathbf{x}_i \mathbf{w} + b) \leq 0$:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i [y_i(\mathbf{x}_i \mathbf{w} + b) - 1], \quad \alpha_i \geq 0. \quad (5)$$

Why $\alpha_i \geq 0$, informally: if a constraint is violated, $y_i(\mathbf{x}_i \mathbf{w} + b) - 1 < 0$, and the term $-\alpha_i[\dots]$ is a positive penalty that grows with α_i . The primal problem is equivalent to $\min_{\mathbf{w}, b} \max_{\alpha \geq 0} L$; the **dual** swaps the order, $\max_{\alpha \geq 0} \min_{\mathbf{w}, b} L$. For the SVM both give the same answer (details in SVM Advanced note, §1).

2.3 Minimize Over $\mathbf{w}, b$

Set the derivatives to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i^T = 0 \quad \Rightarrow \quad \boxed{\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i^T} \quad (6)$$

$$\frac{\partial L}{\partial b} = - \sum_i \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^N \alpha_i y_i = 0. \quad (7)$$

- Equation 6 is the **duality** we have seen before (dual perceptron, kernel ridge regression): $\mathbf{w}$ is a combination of the datapoints, with coefficients $\alpha_i y_i$.
- Equation 7 is a *balance* condition: the total weight $\sum \alpha_i$ on positive points equals the total weight on negative points (picture forces pushing on the hyperplane from both sides, in equilibrium).

2.4 The Dual Problem

Substitute $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i^T$ back into L . Expanding,

$$L = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \underbrace{\sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{w}}_{=\mathbf{w}^T \mathbf{w}} - b \underbrace{\sum_i \alpha_i y_i}_{=0} + \sum_i \alpha_i = \sum_i \alpha_i - \frac{1}{2} \mathbf{w}^T \mathbf{w}, \quad (8)$$

and $\mathbf{w}^T \mathbf{w} = \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j^T$. The result depends only on α :

$$\boxed{\max_{\alpha} W(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j^T \quad \text{s.t.} \quad \alpha_i \geq 0, \quad \sum_{i=1}^N \alpha_i y_i = 0} \quad (9)$$

One variable per training point; the features appear only through the dot products $\mathbf{x}_i \mathbf{x}_j^T$.

2.5 Support Vectors

At the optimum, every term of the Lagrangian sum is zero (**complementary slackness**, one of the KKT conditions, SVM Advanced note, §2):

$$\alpha_i [y_i (\mathbf{x}_i \mathbf{w} + b) - 1] = 0 \quad \text{for every } i. \quad (10)$$

So for each point, either $\alpha_i = 0$, or $y_i (\mathbf{x}_i \mathbf{w} + b) = 1$ (it lies exactly on a margin hyperplane). The points with $\alpha_i > 0$ are the **support vectors**.

Key point: Only support vectors appear in $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i^T$. All other training points could be deleted without changing the classifier. Typically the support vectors are a small fraction of the data, so the trained SVM is a *sparse* combination of training points — unlike kernel ridge regression, where every α_i is nonzero.

Connection to active learning. Recall active learning (HW2): from a pool of points, select the few that are most useful — those near the decision boundary — and train on them. The SVM does something similar, but *inside* a single training run:

- It figures out which training points are useful: the support vectors, the points on (or inside) the margin, i.e. near the boundary.
- The influence of each point on the classifier is explicit: α_i is the “force” with which point i pushes on the hyperplane (and $\sum_i \alpha_i y_i = 0$ says these forces balance).
- The theory guarantees that points with $\alpha_i = 0$ are irrelevant: removing them and retraining gives the same classifier.
- The difference from active learning: there is no staged “first select, then train.” Selection and training happen together; which points end up with $\alpha_i > 0$ is only known once the optimization is done. (And the SVM still needs *labels* for all training points, whereas active learning’s purpose is to avoid labeling most of them.)

3 Soft Margin: Non-Separable Data

Real data is rarely separable, and even when it is, one outlier can force a narrow margin (Figure 1): the hard-margin SVM must keep *every* point outside the band, so a single unusual point can tilt the hyperplane and shrink the band.

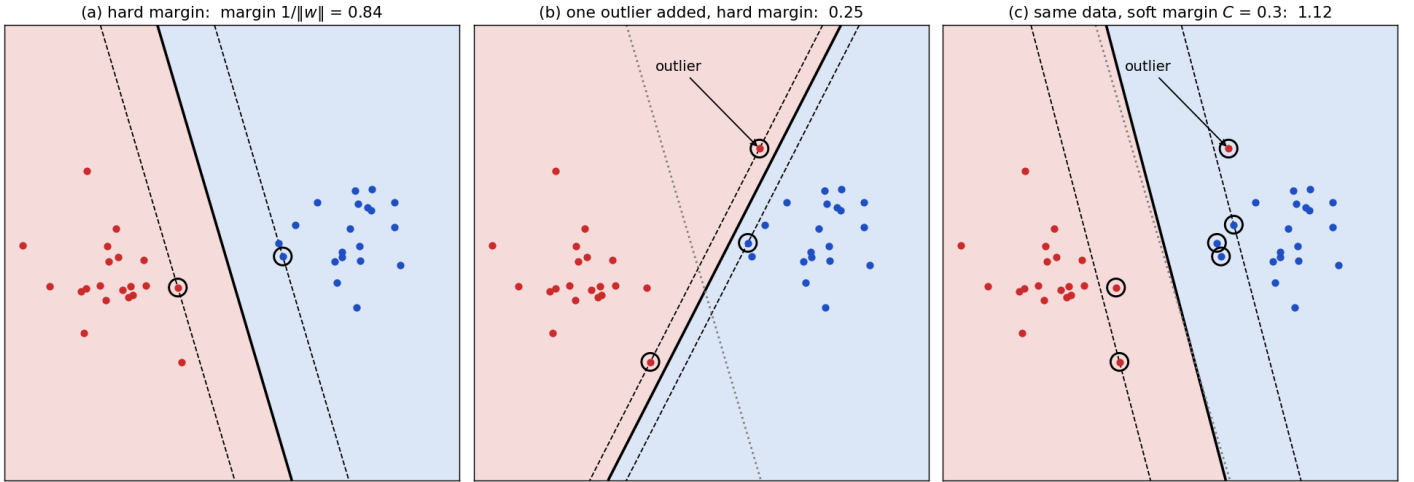


Figure 1: One outlier and the hard margin. (a) Separable data: the hard-margin SVM, margin $1/\|\mathbf{w}\| = 0.84$, two support vectors (circled). (b) The same data plus one red point near the blue class: still separable, so the hard margin must accommodate it; the hyperplane tilts by about 44° and the margin drops to 0.25. Dotted gray: the boundary of (a). (c) The soft margin defined below ($C = 0.3$) instead lets the outlier violate the margin (slack $\xi \approx 2.2$, misclassified) and recovers almost the boundary of (a), with a wide margin.

Allow each point to violate its constraint by a **slack** $\xi_i \geq 0$, and pay for it:

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \quad \text{s.t.} \quad y_i(\mathbf{x}_i \mathbf{w} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \quad (11)$$

- $\xi_i = 0$: the point is outside the band, on the correct side. $0 < \xi_i \leq 1$: inside the band but still correctly classified. $\xi_i > 1$: misclassified.

- $C > 0$ trades the two goals: **large** C punishes violations heavily (narrow margin, fits the training data closely, close to the hard margin); **small** C tolerates violations to get a wider margin. C is a regularization hyperparameter, chosen on a validation set. (The penalty $\sum_i \xi_i$ is an L_1 penalty on the violations.)

The dual (derivation in SVM Advanced note, §3) is *the same* as Equation 9, except that each α_i is now also bounded above by C :

$$\boxed{\max_{\alpha} W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j^T \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_i \alpha_i y_i = 0} \quad (12)$$

Three types of points (from the KKT conditions, SVM Advanced note, §2):

α_i	position	role
$\alpha_i = 0$	outside the band, correct side: $y_i f(\mathbf{x}_i) \geq 1$	not a support vector
$0 < \alpha_i < C$	exactly on the margin: $y_i f(\mathbf{x}_i) = 1$	support vector (“free”)
$\alpha_i = C$	inside the band or misclassified: $y_i f(\mathbf{x}_i) \leq 1$	support vector (“at bound”)

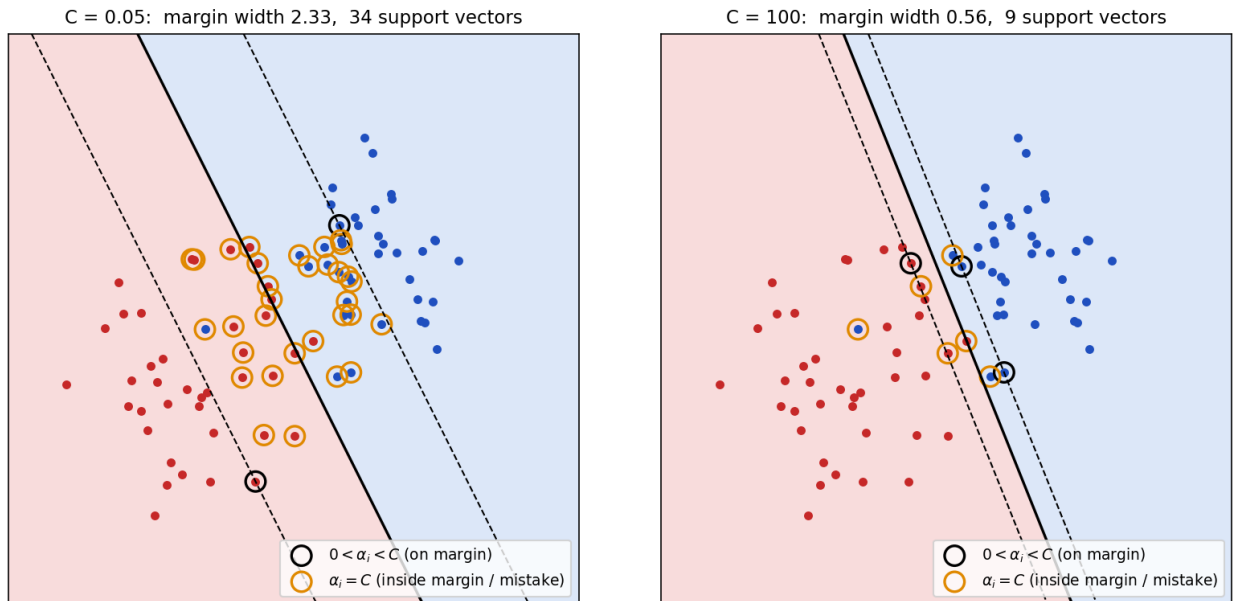


Figure 2: Linear soft-margin SVM on overlapping classes. Solid: decision boundary; dashed: margins $f(\mathbf{x}) = \pm 1$. Black circles: free support vectors ($0 < \alpha_i < C$), exactly on the margin. Orange circles: support vectors at the bound ($\alpha_i = C$), inside the band or misclassified. All other points have $\alpha_i = 0$. Small C (left): wide margin, many violations, many support vectors. Large C (right): narrow margin, few violations.

4 Kernels: The SVM in Feature Space

4.1 The Dual Needs Only K

Look again at the dual (12): the data enters only through the dot products $\mathbf{x}_i \mathbf{x}_j^T$. By the kernel trick (Kernels note), replace them with any valid kernel $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_j)^T$:

$$\max_{\alpha} W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_i \alpha_i y_i = 0 \quad (13)$$

This is *exactly* the soft-margin SVM on the mapped data $\Phi(\mathbf{x})$: a maximum-margin hyperplane in the kernel's feature space, found without ever computing Φ . From here on the whole training and prediction pipeline works with the $N \times N$ kernel matrix.

4.2 Predicting Without $\mathbf{w}$

In feature space $\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)^T$, which may be infinite-dimensional (Gaussian kernel), so we never form it. A prediction needs only kernel values against the support vectors (Kernels note, §7 draws this computation as a diagram):

$$f(\mathbf{z}) = \Phi(\mathbf{z}) \mathbf{w} + b = \sum_{i: \alpha_i > 0} \alpha_i y_i K(\mathbf{x}_i, \mathbf{z}) + b, \quad h(\mathbf{z}) = \text{sign } f(\mathbf{z}) \quad (14)$$

The intercept b . A free support vector j ($0 < \alpha_j < C$) lies exactly on the margin, $y_j f(\mathbf{x}_j) = 1$, i.e. $f(\mathbf{x}_j) = y_j$ (since $y_j^2 = 1$). Solving for b :

$$b = y_j - \sum_i \alpha_i y_i K_{ij}, \quad (15)$$

averaged over all free support vectors for numerical stability. (SMO also maintains b as it goes, §5.) With the linear kernel one can still form $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i^T$ explicitly.

Key point: Compare the three kernelized classifiers we have: **kernel ridge regression** ($\alpha = (K + \lambda I)^{-1} \mathbf{y}$, every $\alpha_i \neq 0$), the **dual perceptron** (α_i = number of mistakes on $\mathbf{x}_i$, any separating hyperplane), and the **SVM** (α from the dual QP: the maximum-margin hyperplane, with $\alpha_i \neq 0$ only for support vectors). All three predict with $\sum_i (\text{weight}_i) K(\mathbf{x}_i, \mathbf{z})$.

4.3 Choosing the Kernel and C

With the Gaussian kernel $K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / \sigma^2)$ there are two hyperparameters, and both control how complex the boundary can be (Figure 3):

- σ (kernel width): small σ means each support vector influences only its immediate neighborhood, so the boundary can bend around individual points; large σ gives smooth boundaries (as $\sigma \rightarrow \infty$ the RBF-SVM behaves more and more like a linear one).
- C : small C tolerates violations (smoother, more support vectors); large C tries to classify every training point correctly.
- Choose both on a validation set, typically on a logarithmic grid (e.g. $C \in \{0.01, 0.1, \dots, 1000\}$). The Kernels note's gallery gives guidance on *which* kernel to try: linear for high-dimensional sparse data, Gaussian as the default for low-dimensional dense data.

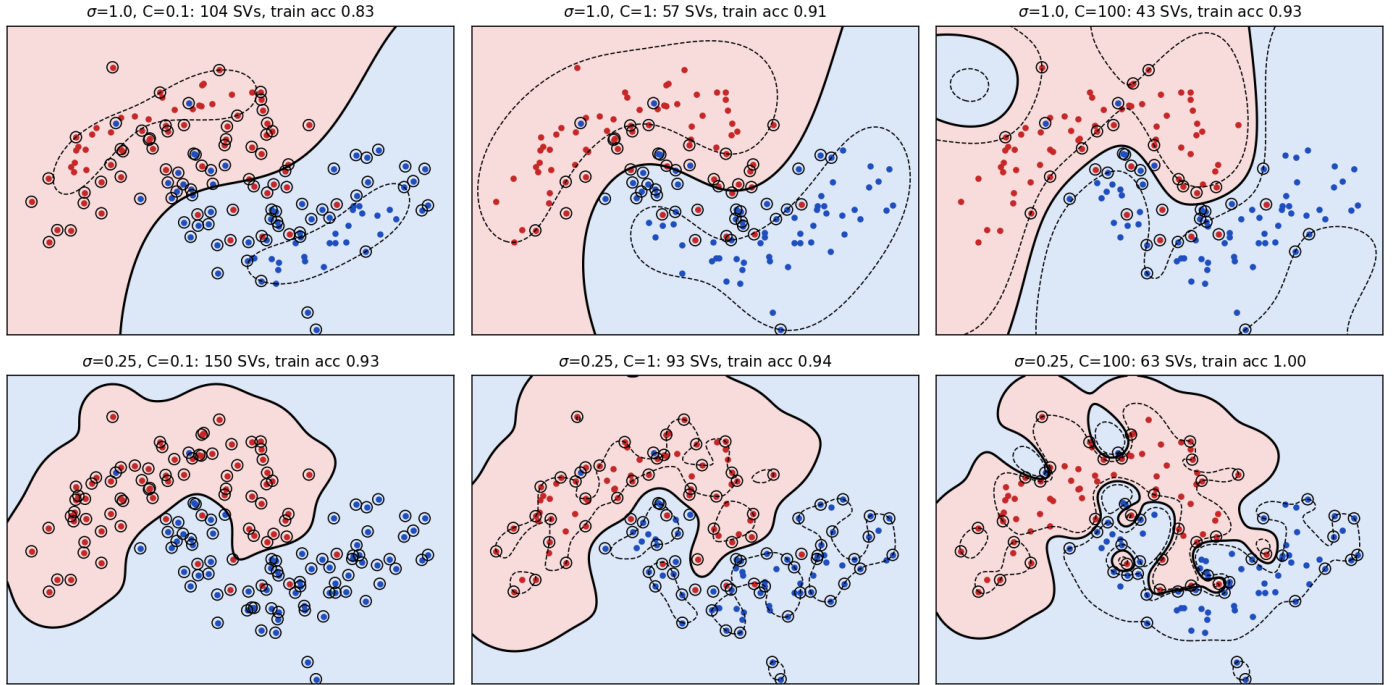


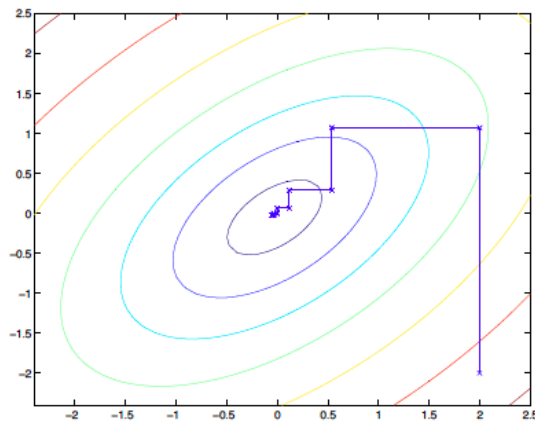
Figure 3: RBF-kernel SVM on the two-moons dataset. Rows: $\sigma = 1$ (top) and $\sigma = 0.25$ (bottom). Columns: $C = 0.1, 1, 100$. Solid: decision boundary; dashed: margins; circled: support vectors. Small σ and large C (bottom right) fit the training data perfectly with a very irregular boundary.

5 Solving the Dual: SMO

5.1 Why a Special Algorithm

The dual (13) is a quadratic program in N variables. A generic QP solver works for small N , but it handles the full $N \times N$ matrix at every step and becomes slow and memory-hungry for large datasets. **SMO** (Sequential Minimal Optimization, Platt 1998) instead solves a sequence of tiny problems, each with a closed-form solution.

Coordinate ascent. To maximize a function of many variables, repeatedly pick one variable, hold all others fixed, and maximize over that one alone. Each step moves parallel to one axis:



Why two variables at a time. For the SVM dual, one variable alone cannot move: the constraint $\sum_i \alpha_i y_i = 0$ gives $\alpha_1 = -y_1 \sum_{i \geq 2} \alpha_i y_i$, so if all others are fixed, α_1 is fixed too. The smallest possible step

changes **two** variables α_i, α_j together, keeping $\alpha_i y_i + \alpha_j y_j$ constant.

5.2 One SMO Step

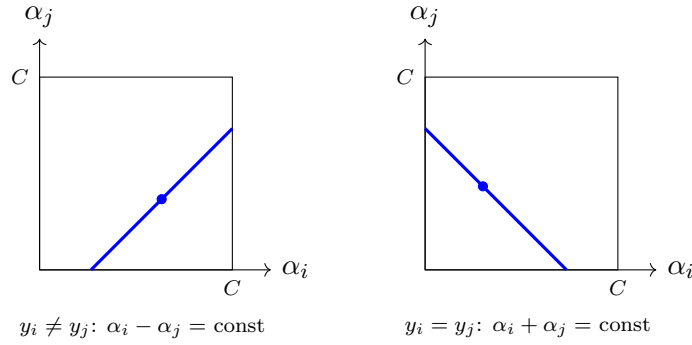
Fix all variables except α_i, α_j . Define the current score and error of each point,

$$f(\mathbf{x}_k) = \sum_l \alpha_l y_l K_{lk} + b, \quad E_k = f(\mathbf{x}_k) - y_k, \quad (16)$$

and

$$\eta = K_{ii} + K_{jj} - 2K_{ij} = \|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_j)\|^2 \geq 0. \quad (17)$$

1. **The pair lives on a line segment.** $\alpha_i y_i + \alpha_j y_j$ is constant, and both variables must stay in $[0, C]$: the pair moves along a line inside the box $[0, C] \times [0, C]$:



So α_j is limited to an interval $[L, H]$:

$$y_i \neq y_j: \quad L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i); \quad (18)$$

$$y_i = y_j: \quad L = \max(0, \alpha_i + \alpha_j - C), \quad H = \min(C, \alpha_i + \alpha_j). \quad (19)$$

2. **Best α_j on the line.** Along the line, W is a quadratic in α_j alone (curvature $-\eta$). Its maximum is at

$$\alpha_j^{new} = \alpha_j + \frac{y_j(E_i - E_j)}{\eta} \quad (20)$$

(derivation in SVM Advanced note, §4). Intuition: the step is driven by the difference in the two points' errors.

3. **Clip** to the segment: $\alpha_j^{new} \leftarrow \min(H, \max(L, \alpha_j^{new}))$.
4. **Update** α_i to keep $\alpha_i y_i + \alpha_j y_j$ constant: $\alpha_i^{new} = \alpha_i + y_i y_j (\alpha_j - \alpha_j^{new})$.
5. **Update** b so that a free support vector among the two lands exactly on the margin:

$$b_1 = b - E_i - y_i(\alpha_i^{new} - \alpha_i)K_{ii} - y_j(\alpha_j^{new} - \alpha_j)K_{ij}, \quad (21)$$

$$b_2 = b - E_j - y_i(\alpha_i^{new} - \alpha_i)K_{ij} - y_j(\alpha_j^{new} - \alpha_j)K_{jj}, \quad (22)$$

$$b = b_1 \text{ if } 0 < \alpha_i^{new} < C; \text{ else } b = b_2 \text{ if } 0 < \alpha_j^{new} < C; \text{ else } b = (b_1 + b_2)/2.$$

Remark. We use Platt's sign convention, $\eta = K_{ii} + K_{jj} - 2K_{ij} \geq 0$. The CS229 Simplified SMO notes define η with the opposite sign (and flip the update and the skip test accordingly). It is the same step; if you consult both, don't mix the two conventions.

5.3 Which Pairs? The Simplified SMO Algorithm

A point needs fixing if it violates its row of the three-types table (§3): in terms of $y_i E_i = y_i f(\mathbf{x}_i) - 1$, a violation is $y_i E_i < -\text{tol}$ with $\alpha_i < C$ (should be pushed up) or $y_i E_i > \text{tol}$ with $\alpha_i > 0$ (should be pushed down). The Simplified SMO (from the CS229 notes) loops over all such points and pairs each one with a randomly chosen second point:

Algorithm: Simplified SMO

1. Input: kernel matrix K ($N \times N$), labels y , C , tolerance tol , max_passes . Initialize $\alpha = \mathbf{0}$, $b = 0$, $\text{passes} = 0$.
2. While $\text{passes} < \text{max_passes}$:
 - (a) $\text{changed} = 0$. For each i : compute E_i . If α_i violates the conditions above:
 - i. pick $j \neq i$ at random; compute E_j ; save old α_i, α_j ;
 - ii. compute L, H ; if $L = H$, continue to the next i ;
 - iii. compute η ; if $\eta \leq 0$, continue;
 - iv. $\alpha_j \leftarrow \alpha_j + y_j(E_i - E_j)/\eta$, clipped to $[L, H]$; if it barely changed ($< 10^{-5}$), continue;
 - v. update α_i , then b ; $\text{changed} \leftarrow \text{changed} + 1$.
 - (b) $\text{passes} \leftarrow \text{passes} + 1$ if $\text{changed} = 0$, else $\text{passes} \leftarrow 0$.
3. Output α, b . Predict with Equation 14.

The full Platt algorithm chooses the pair more cleverly (SVM Advanced note, §6).

5.4 Worked Example

Two points: $\mathbf{x}_1 = (1, 0)$, $y_1 = +1$ and $\mathbf{x}_2 = (-1, 0)$, $y_2 = -1$; linear kernel ($K_{11} = K_{22} = 1$, $K_{12} = -1$); $C = 10$; start from $\alpha = (0, 0)$, $b = 0$.

- All scores are 0, so $E_1 = 0 - 1 = -1$, $E_2 = 0 - (-1) = 1$. Both points violate the conditions.
- Pair $(i, j) = (1, 2)$: $y_1 \neq y_2$, so $L = \max(0, 0) = 0$, $H = \min(10, 10) = 10$. $\eta = 1 + 1 - 2(-1) = 4$.
- $\alpha_2^{\text{new}} = 0 + \frac{(-1)(-1-0)}{4} = 0.5$ (inside $[L, H]$, no clipping). $\alpha_1^{\text{new}} = 0 + (1)(-1)(0 - 0.5) = 0.5$.
- Both are free, so $b = b_1 = 0 - (-1) - (1)(0.5)(1) - (-1)(0.5)(-1) = 1 - 0.5 - 0.5 = 0$.
- Result: $\mathbf{w} = 0.5 \mathbf{x}_1^T - 0.5 \mathbf{x}_2^T = (1, 0)^T$, $b = 0$: the boundary is the vertical axis, halfway between the points; margin $1/\|\mathbf{w}\| = 1$ (half their distance); both points are support vectors with $f = \pm 1$. Already optimal, so the next passes change nothing. ✓

Adding a third point $\mathbf{x}_3 = (2, 1)$, $y_3 = +1$ (farther from the boundary) does not change the solution: $\alpha_3 = 0$, not a support vector.

6 Multiclass SVMs

The SVM is a binary classifier. For k classes, combine several:

- **One-vs-rest:** train k SVMs, SVM c separating class c (label $+1$) from all other classes (-1). Each one answers a yes/no question, “is $\mathbf{z}$ in class c ?”, via $\text{sign } f_c(\mathbf{z})$. Ideally exactly one says yes; in practice a test point can get several “yes” answers, or none. A common decision rule is to predict the class with the largest score, $\arg \max_c f_c(\mathbf{z})$, even if that score is negative.
- **One-vs-one (all pairs):** train $\binom{k}{2}$ SVMs, one per pair of classes, each on the points of those two classes only; each casts one vote, and the class with the most votes wins (ties are possible and must be broken somehow). This is what `sklearn.svm.SVC` does internally.
- ECOC codes (Boosting note) work with any binary classifier, including SVMs.

HW3’s THEORY Problem 6 asks you to think about the regions where the yes/no answers of one-vs-rest are ambiguous.

7 Implementation Notes (HW3)

- **Problem 1, Parts A–B** (library SVM): `sklearn.svm.SVC` with `kernel`, `C`, and `gamma=1/\sigma^2`; tune C and the kernel parameters on a validation split. Standardize features first (distances and dot products depend on scale). For Digits, `SVC` is already multiclass (one-vs-one); wrapping it in `OneVsRestClassifier` gives one-vs-rest.
- **Problem 1, Parts C–D** (Simplified SMO from scratch): write the solver against a precomputed kernel matrix K only, so the same code runs with the linear kernel (Part C) and the RBF kernel (Part D). Cap the total number of sweeps (e.g. 300): on noisy data the “max_passes without change” stopping rule may never trigger.
- **Checks:** with the linear kernel, $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i^T$ should match `SVC(kernel='linear')`’s `coef_` reasonably well at the same C : same direction (cosine similarity close to 1) and similar accuracy. An exact match also of the magnitude requires full convergence, which a sweep cap may not reach on non-separable data (it did on a well-separated two-blob dataset: three decimals). Another check is the set of support vectors (points with $\alpha_i > 0$), which should largely coincide with `SVC`’s `support_`. With the RBF kernel, compare accuracies.
- **Common bugs:** labels in $\{0, 1\}$ instead of $\{-1, +1\}$; recomputing E_j before saving the old α ’s; forgetting to clip; picking $j = i$; mixing the two η sign conventions; computing b only at the end with the wrong formula.

Further reading (optional): the **SVM Advanced note** has the derivations behind this note (Lagrange duality, the KKT conditions, the soft-margin dual, the SMO update), the full Platt SMO, solving the dual with a QP library, the SVM as hinge-loss minimization (Pegasos), support vector regression, and the VC-dimension view. None of it is needed for HW3’s required problems.