Five classic ideas worth knowing

The last two lectures were about **which columns** the model sees — building them ([26]) and cutting them ([27]). This one is about **how it draws the boundary** once it has them. Trees and boosting ([17]–[20]) win most tabular problems today, but five **classic** methods still show up everywhere — in interviews, in libraries, and as the *ideas* behind modern tools. Each has **one beautiful, transferable idea**.

Family	**Method → the idea**
Learn by **similarity**	KNN → "you are like your neighbors"
Model how classes are **generated**	Naive Bayes, LDA/QDA → Bayes' rule
Margins & kernels	**SVM** → widest street + the **kernel trick**
	Gaussian processes → kernels, now with *uncertainty*

SVM is the focus — we derive it. The other four are ideas to *recognize*, intuition-first.

Outline

Learn by similarity: KNN

Generative classifiers: Naive Bayes, LDA/QDA

Margins & kernels: Support Vector Machines

Gaussian processes

Learn by similarity

Predict a point the way its nearest neighbors voted — no training, all the work at prediction time.

K-nearest neighbors: lazy learning

No training-time optimization at all. To predict a new point:

1. find its k **nearest** training points (by distance);
2. **vote** (classification) or **average** (regression).

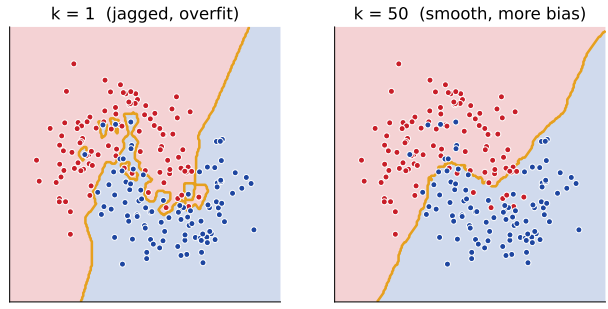
The cost is moved, not removed: *zero* training, but *expensive* prediction — every query scans the data ($O(n \cdot p)$).

k is the bias-variance knob

The boundary is **local and jagged** (a Voronoi mosaic).

- $k = 1$: zero training error, **overfits** noise.
- large k: smoother, **more bias**.
- choose k by **CV** (callback: overfitting/CV lecture).

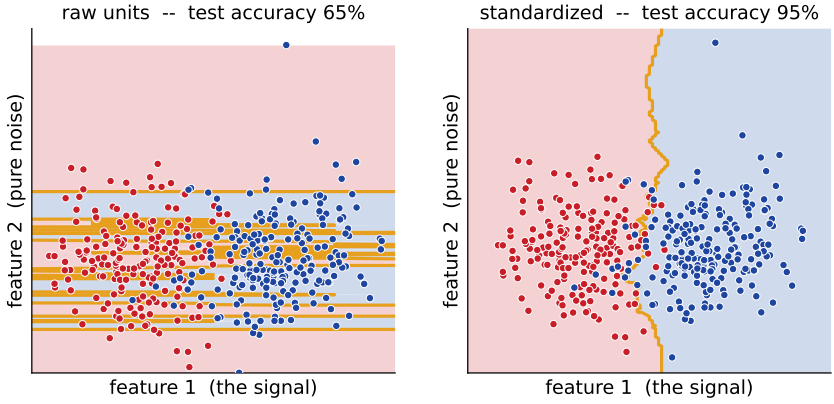
Predict-first: what does $k = 1$ do on noisy data? *(Memorizes it — islands around single points.)*



KNN: three gotchas

- **Scale first.** Distances are dominated by large-range features (callback: the preprocessing lecture) — standardize.
- **Curse of dimensionality.** In high dimensions everything is far from everything; "nearest" stops meaning much.
- **Inference cost.** $O(n \cdot p)$ per query $\rightarrow$ speed it up with kd-/ball-trees, or approximate nearest neighbors (ANN, FAISS) at scale.

“Scale first” is a prerequisite, not advice



Feature 2 is **pure noise** — it carries no information about the class whatsoever. But its numbers are 100× bigger, so it swallows the distance and KNN ends up slicing along the useless axis. **Test accuracy 65%.** One `StandardScaler` later, on the very same data: **95%**. Nothing was added — a column was merely stopped from shouting.

KNN: where the idea still wins

Transferable idea — similarity search. Nearest neighbors in an **embedding space** is exactly **retrieval**: vector search, RAG, recommenders ("users like you also…").

Also handy for: **KNN imputation**, **anomaly detection** (far from all neighbors), and a **quick baseline** that surprisingly often competes.

Model how each class is generated

Learn what each class *looks like*, then let Bayes' rule turn that into a decision.

Naive Bayes: from words to a decision

Is this SMS spam? We want $P(\text{spam} \mid \text{words})$. Bayes' rule:

$$\underbrace{P(y \mid \mathbf{x})}_{\text{posterior}} \propto \underbrace{P(\mathbf{x} \mid y)}_{\text{likelihood}} \underbrace{P(y)}_{\text{prior}}, \qquad \text{predict } \arg\max_y P(y \mid \mathbf{x}).$$

Generative: instead of drawing a boundary, model how each class *produces* its features, then compare.

The "naive" assumption (and its variants)

Estimating the full $P(\mathbf{x} \mid y)$ is hard. **Naive** shortcut: features are **conditionally independent given the class**, so

$$P(\mathbf{x} \mid y) = \prod_{j=1}^{p} P(x_j \mid y).$$

Obviously false ("free" and "money" co-occur in spam) — but cheap.

Variants by feature type: **Gaussian**NB (continuous), **Multinomial**/**Bernoulli**NB (word counts / presence). **Laplace smoothing** avoids a single unseen word zeroing the whole product.

By hand: computing $P(y \mid \mathbf{x})$ for one message

Message: `"free money"`. Toy counts estimated from the training set:

	spam	**ham**
$P(y)$	0.4	0.6
$P(\texttt{free} \mid y)$	0.4	0.02
$P(\texttt{money} \mid y)$	0.3	0.05

Step 1 — score each class: prior $\times$ the two likelihoods.

$$s_{\text{spam}} = 0.4 \cdot 0.4 \cdot 0.3 = 0.048, \qquad s_{\text{ham}} = 0.6 \cdot 0.02 \cdot 0.05 = 0.0006.$$

Step 2 — divide by the total. That total *is* $P(\mathbf{x})$ — the denominator Bayes' rule quietly drops when you only need the winner:

$$P(\text{spam} \mid \mathbf{x}) = \frac{0.048}{0.048 + 0.0006} = \frac{0.048}{0.0486} = \mathbf{0.988}, \quad P(\text{ham} \mid \mathbf{x}) = 0.012.$$

Two different questions. For *which class* you never need Step 2 — the ratio 0.048 : 0.0006 already says spam by 80:1. For *how confident*, you must normalize. *(In code the products are summed in logs, so a long message cannot underflow to zero.)*

One unseen word can veto an entire class

Suppose `crypto` never appeared in a *ham* message in training. Then $P(\texttt{crypto} \mid \text{ham}) = 0/n = 0$, and for *any* message containing it:

$$s_{\text{ham}} \;=\; 0.6 \cdot 0.02 \cdot 0.05 \cdot \mathbf{0} \;=\; \mathbf{0}.$$

Exactly zero — no matter how ham-like every other word in the message was. A product has no mercy: one missing count deletes the whole class from consideration.

Laplace (add-one) smoothing: pretend every word was seen once more than it actually was, $P(w \mid y) = \dfrac{\text{count} + 1}{\text{total} + |V|}$. With a 10,000-word vocabulary and 5,000 ham words, $P(\texttt{crypto} \mid \text{ham}) = \frac{1}{15{,}000} \approx 0.000067$ instead of 0 — vanishingly small, but **still in the running**.

Why it works despite a false assumption

Predict-first: the independence assumption is wrong — `free` and `money` really do co-occur. Shouldn't the classifier fail?

Why it works despite a false assumption

Predict-first: the independence assumption is wrong — `free` and `money` really do co-occur. Shouldn't the classifier fail?

No. We only need the **argmax** to be right, not the probabilities — the estimated numbers are often badly off, but the *ranking* of classes survives.

Multiplying p simple 1-D estimates **dodges the curse of dimensionality**, needs **little data**, and is **very fast**.

This is exactly why that **0.988** is **not** a trustworthy 98.8%. Counting each word as independent evidence double-counts correlated words, so the products get pushed toward 0 and 1. The **order** is right; the **number** is overconfident.

The rule is optimal. The estimate is not.

Suppose you *knew* the true $P(\mathbf{x} \mid y)$ and $P(y)$. Then predicting $\arg\max_y P(y \mid \mathbf{x})$ is **provably the best any classifier can do** — the **Bayes classifier**. No model, however clever, beats it.

Its error is still not zero, because two classes can genuinely produce the same $\mathbf{x}$:

$$\text{Bayes error} \;=\; 1 - \mathbb{E}_{\mathbf{x}}\Big[\max_y P(y \mid \mathbf{x})\Big].$$

That is the **irreducible floor** — the same noise term from the bias-variance decomposition ([06]). You will *see* it later in this section, as the overlap where the red and blue curves cross.

So Naive Bayes, LDA and QDA all apply the **same optimal rule**. They differ only in how they estimate $P(\mathbf{x} \mid y)$: diagonal Gaussian, shared full Gaussian, per-class Gaussian. **When they lose, Bayes' rule never failed** — the density estimate did.

Generative vs discriminative

Generative (NB, LDA/QDA)

- models $P(\mathbf{x} \mid y)$ — "how each class looks";
- Bayes' rule $\rightarrow$ boundary.

Discriminative (logreg, SVM)

- models the boundary / $P(y \mid \mathbf{x})$ directly;
- no model of the features.

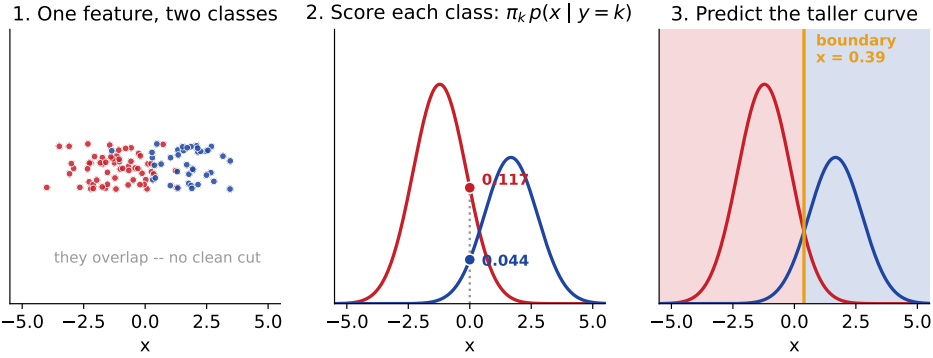
Naive Bayes gives a **score, not a calibrated probability** (callback: the calibration lecture) — its confident-looking numbers are usually miscalibrated.

LDA / QDA: Gaussian blobs, then Bayes

LDA (*linear discriminant analysis*) and **QDA** (*quadratic* discriminant analysis) model each class as a **Gaussian blob** in feature space; Bayes' rule turns the blobs into a boundary. This is Naive Bayes' **richer cousin** — it *allows feature correlations* (a full covariance matrix) instead of assuming independence.

Same Bayes formula as NB — **only the covariance assumption changes**.

The whole idea, in one dimension



1. Two overlapping classes. No cut separates them.
2. **Fit one Gaussian per class**, scaled by that class's share π_k. At $x = 0$: **0.117** red against **0.044** blue.
3. **Predict whoever is taller** — the boundary is just where the curves cross.

Red has 60 points to blue's 40, so π_k pushes the crossing *right* of the midpoint — the commoner class gets more territory.

That is all LDA and QDA do. The rest of this section is only about what "a bell in p dimensions" means.

First, in two dimensions: what Σ^{-1} measures

One class, mean $\boldsymbol{\mu} = (0, 0)$, and a blob **four times wider than it is tall**:

$$\Sigma = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix} \qquad \Rightarrow \qquad \Sigma^{-1} = \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{pmatrix}.$$

Predict-first: $\mathbf{x}_A = (2, 0)$ and $\mathbf{x}_B = (0, 2)$ are both at Euclidean distance 2 from the mean. Which is the better fit to this class?

First, in two dimensions: what Σ^{-1} measures

One class, mean $\boldsymbol{\mu} = (0, 0)$, and a blob **four times wider than it is tall**:

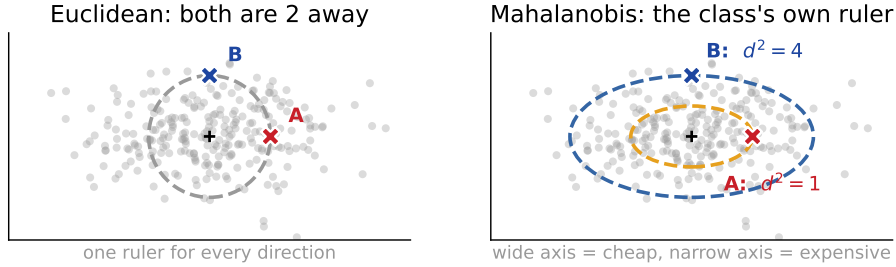
$$\Sigma = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix} \qquad \Rightarrow \qquad \Sigma^{-1} = \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & 1 \end{pmatrix}.$$

Predict-first: $\mathbf{x}_A = (2, 0)$ and $\mathbf{x}_B = (0, 2)$ are both at Euclidean distance 2 from the mean. Which is the better fit to this class?

$$(\mathbf{x}_A - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}_A - \boldsymbol{\mu}) = \tfrac{4}{4} = 1, \qquad (\mathbf{x}_B - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}_B - \boldsymbol{\mu}) = \tfrac{4}{1} = 4.$$

Same raw distance, **four times** the penalty. Σ^{-1} divides each direction by how much the class *naturally varies* there: 2 units along a wide axis is unremarkable, 2 units along a narrow one is not. That is all **Mahalanobis distance** means — and it is the only new idea in the formula on the next slide.

The same two points, drawn



Left: one ruler for every direction — A and B are both 2 away, so Euclidean distance calls them equally typical. **Right:** the dashed rings are the class's *own* contours of equal distance. A lands on the inner ring, B on the outer one. Same points, same blob — the only thing that changed is **which ruler** we measured with.

From a blob to a score: the discriminant

Write the Gaussian for class k (mean $\boldsymbol{\mu}_k$, covariance Σ_k, prior $\pi_k = P(y{=}k)$):

$$p(\mathbf{x} \mid y{=}k) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\Big(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\Big).$$

Take log of $\pi_k\, p(\mathbf{x} \mid y{=}k)$ and drop shared constants $\rightarrow$ the **discriminant score** (predict $\arg\max_k \delta_k$):

$$\delta_k(\mathbf{x}) = -\tfrac{1}{2}\log|\Sigma_k| \;-\; \tfrac{1}{2}\underbrace{(\mathbf{x}-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)}_{(\text{Mahalanobis distance to } \boldsymbol{\mu}_k)^2} + \log\pi_k.$$

In words: **assign x to the nearest class blob** — distance measured in the class's own stretched metric Σ_k^{-1}, plus a prior nudge $\log\pi_k$.

Why shared Σ makes the boundary linear

Expand the distance term:

$$(\mathbf{x}-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) = \underbrace{\mathbf{x}^\top \Sigma_k^{-1}\mathbf{x}}_{\text{quadratic in } \mathbf{x}} - 2\boldsymbol{\mu}_k^\top \Sigma_k^{-1}\mathbf{x} + \boldsymbol{\mu}_k^\top \Sigma_k^{-1}\boldsymbol{\mu}_k.$$

QDA (Σ_k per class): the $\mathbf{x}^\top \Sigma_k^{-1}\mathbf{x}$ term *differs* by class $\Rightarrow$ δ_k stays **quadratic** $\Rightarrow$ the boundary is a **curve**.

LDA ($\Sigma_k = \Sigma$ shared): $\mathbf{x}^\top \Sigma^{-1}\mathbf{x}$ and $\log|\Sigma|$ are the *same* for every class $\Rightarrow$ they **cancel**.

What survives for LDA is **linear** in $\mathbf{x}$ — a hyperplane:

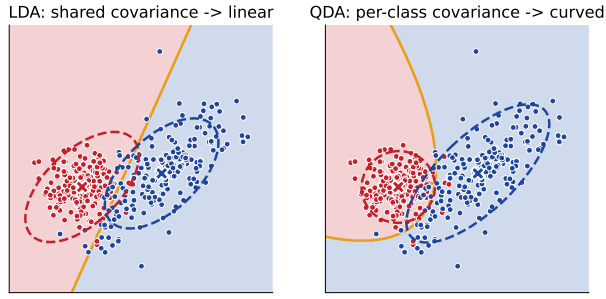
$$\delta_k(\mathbf{x}) = (\Sigma^{-1}\boldsymbol{\mu}_k)^\top \mathbf{x} - \tfrac{1}{2}\boldsymbol{\mu}_k^\top \Sigma^{-1}\boldsymbol{\mu}_k + \log \pi_k = \mathbf{w}_k^\top \mathbf{x} + b_k.$$

The cancellation *is* the reason: shared covariance kills the quadratic term, leaving a line. *And look at the shape:* $\mathbf{w}^\top \mathbf{x} + b$ is **logistic regression's** boundary, exactly. LDA got there by modelling $p(\mathbf{x} \mid y)$, logreg by modelling $P(y \mid \mathbf{x})$ directly — two different routes, the same form.

LDA vs QDA: one knob, the covariance

- **LDA:** *shared* covariance across classes $\Rightarrow$ a **linear** boundary (fewer parameters, more bias).
- **QDA:** *per-class* covariance $\Rightarrow$ a **quadratic**, curved boundary (more parameters, more variance).

One covariance knob: Gaussian NB (*diagonal* Σ) $\subset$ LDA (shared full Σ) $\subset$ QDA (per-class Σ_k). Fewer covariance parameters = more bias.



LDA is also supervised dimensionality reduction

Beyond classifying, **LDA** finds the axes that best **separate the classes** — project onto $\leq (\#\text{classes} - 1)$ dimensions.

PCA vs LDA: PCA (*principal component analysis*, coming in the dimensionality-reduction lecture) is *unsupervised* — it keeps the directions of most **variance**. LDA is *supervised* — it keeps the directions of most **class separation**.

Predict-first: on a 2-class blob stretched diagonally, which axis would each pick?

LDA is also supervised dimensionality reduction

Beyond classifying, **LDA** finds the axes that best **separate the classes** — project onto $\leq (\#\text{classes} - 1)$ dimensions.

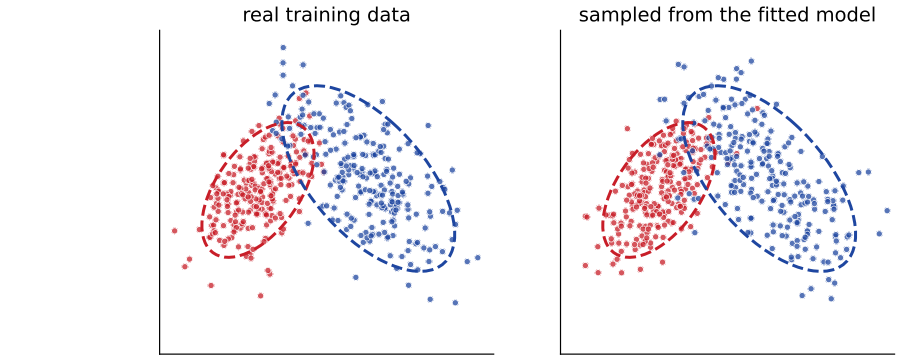
PCA vs LDA: PCA (*principal component analysis*, coming in the dimensionality-reduction lecture) is *unsupervised* — it keeps the directions of most **variance**. LDA is *supervised* — it keeps the directions of most **class separation**.

Predict-first: on a 2-class blob stretched diagonally, which axis would each pick?

PCA → the long (high-variance) axis; LDA → the axis that *separates the two colors*, even if it is the short one.

Strong low-data baseline; assumes roughly Gaussian features.

Generative means you can *generate*



A discriminative model (logreg, SVM) only answers *"which side?"*. A generative one carries a description of how each class *looks*, so you can **run it backwards** and draw new data. Naive Bayes too: sample words from $P(w \mid \text{spam})$ and out comes a fake spam message. **This is the idea behind GANs and diffusion** — learn $P(\mathbf{x})$ well enough that samples from it convince.

Margins & kernels

The centerpiece: the widest street, and a trick that bends it around any shape — derived, not asserted.

Where it came from: Vapnik, 1963 to 2012

Year	Who	What
1963	Vapnik & Chervonenkis	"Generalized Portrait" — the maximum-margin idea. Linear only. Moscow.
1964	Aizerman, Braverman & Rozonoer	kernels, under the name "potential functions"
1974	Vapnik & Chervonenkis	statistical learning theory — *why* a wide margin should generalize
1992	Boser, Guyon & Vapnik	margin + kernel trick = **SVM** (at the COLT conference)
1995	Cortes & Vapnik	"Support-Vector Networks" — the **soft margin**; this is the version in `sklearn`
2012	Krizhevsky, Sutskever & Hinton	**AlexNet** wins ImageNet: top-5 error **15.3%** against **26.2%** for second place

Before 1992, the answer to a hard pattern-recognition problem was a **backpropagation neural net** — LeCun's *LeNet* on handwritten digits (1989), built at the same AT&T Bell Labs where Vapnik was then working. SVM became competitive there *and* carried a theory of why, so for some fifteen years kernel methods were the respectable choice while neural nets were unfashionable and hard to train. **Then the irony of 2012:** every ImageNet winner *before* AlexNet was hand-engineered features fed into an **SVM**. Deep learning did not dethrone some vague "classical ML" — it dethroned this exact pipeline.

Maximum margin: the widest street

Many lines separate these classes. SVM picks the one with the **widest margin** — the largest "safety street" around the boundary, so new points are least likely to cross.

Support vectors = the few points touching the margin edge. *Only they* determine the boundary — move any other point and nothing changes.

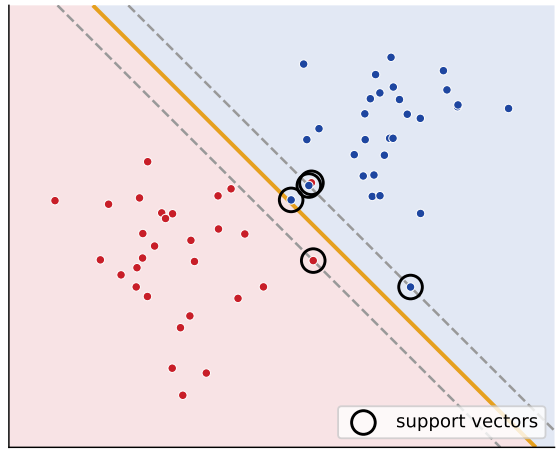
Predict-first: of several separating lines, which generalizes best?

Maximum margin: the widest street

Many lines separate these classes. SVM picks the one with the **widest margin** — the largest "safety street" around the boundary, so new points are least likely to cross.

Support vectors = the few points touching the margin edge. *Only they* determine the boundary — move any other point and nothing changes.

Predict-first: of several separating lines, which generalizes best? *(The widest-margin one.)*



From "widest street" to a quadratic program

Decision function $f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0$, predict $\text{sign}(f)$. The signed distance of a point is $d(f, \mathbf{x}^{(i)}) = y^{(i)} f(\mathbf{x}^{(i)}) / \|\boldsymbol{\theta}\|$.

Maximize the smallest distance M (the **m**argin):

$$\max_{\boldsymbol{\theta}, \theta_0} M \quad \text{s.t.} \quad \frac{y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0)}{\|\boldsymbol{\theta}\|} \geq M.$$

Rescaling $(\boldsymbol{\theta}, \theta_0) \to (c\boldsymbol{\theta}, c\theta_0)$ leaves the *same* hyperplane, so that scale is a free choice — fix it as $\|\boldsymbol{\theta}\| = 1/M$. The constraint becomes $y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) \geq 1$, and $M = 1/\|\boldsymbol{\theta}\|$:

$$\min_{\boldsymbol{\theta}, \theta_0} \ \tfrac{1}{2}\|\boldsymbol{\theta}\|^2$$

$$\text{s.t. } y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) \geq 1 \ \forall i.$$

Maximizing $1/\|\boldsymbol{\theta}\|$ = minimizing $\frac{1}{2}\|\boldsymbol{\theta}\|^2$. A **convex quadratic program** (the *primal*); margin width $= 2/\|\boldsymbol{\theta}\|$.

Support vectors: sparse in the data

The points with $y^{(i)} f(\mathbf{x}^{(i)}) = 1$ sit *exactly* on the margin edge — these are the **support vectors**.

- Delete a *non*-support-vector $\rightarrow$ the solution is **unchanged**.
- Delete a support vector $\rightarrow$ the boundary **moves**.

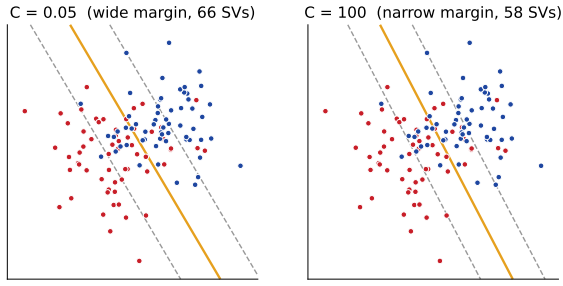
That is why it is a *Support Vector* Machine: the model is **sparse in the data** — it depends only on a handful of points. (We will see this fall out of the dual, algebraically.)

Soft margin: real data isn't cleanly separable

On non-separable data the hard-margin constraints are **contradictory** (empty feasible region). Allow **slack** $\xi_i \geq 0$ — points may sit inside/across the margin:

$$\min_{\boldsymbol{\theta},\theta_0,\boldsymbol{\xi}} \ \tfrac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t. } y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) \geq 1 - \xi_i.$$



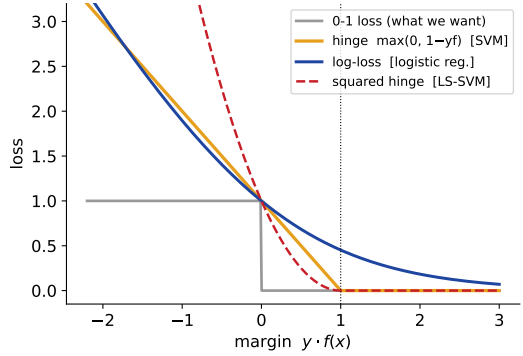
C **is a regularization knob** (bias-variance): large C = punish violations, narrow margin (overfit); small C = wider margin, more bias. $C \to \infty$ recovers the hard margin. *Careful:* C runs **opposite** to Ridge/Lasso's λ — large C means *weak* regularization. (Same convention as `LogisticRegression(C=...)`.)

SVM *is* regularized ERM with the hinge loss

At the optimum, $\xi_i = \max(0,\, 1 - y^{(i)} f_i)$, so the soft-margin SVM is just

$$\min_{\boldsymbol{\theta},\theta_0} \; \tfrac{1}{2}\|\boldsymbol{\theta}\|^2 + C \sum_{i=1}^{n} \max(0,\, 1 - y^{(i)} f_i).$$

That is L_2-**regularized ERM** with the **hinge loss** — the same recipe as Ridge and regularized logistic regression ([07]), just a different loss.



Swap the loss: hinge → **log-loss** gives *regularized logistic regression* — SVM and logreg are the **same recipe**, different loss. The hinge's *flat* part is what creates support vectors (smooth losses have none).

Where the dual comes from

Attach a multiplier $\alpha_i \geq 0$ to each margin constraint and fold them into one **Lagrangian**:

$$L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\alpha}) = \tfrac{1}{2}\|\boldsymbol{\theta}\|^2 \; - \; \sum_{i=1}^{n} \alpha_i \big[y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) - 1 \big].$$

At the optimum the derivatives in $\boldsymbol{\theta}$ and θ_0 vanish:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = 0 \;\Rightarrow\; \boldsymbol{\theta} = \sum_{i=1}^{n} \alpha_i y^{(i)} \mathbf{x}^{(i)}, \qquad \frac{\partial L}{\partial \theta_0} = 0 \;\Rightarrow\; \sum_{i=1}^{n} \alpha_i y^{(i)} = 0.$$

The first one is the whole story: $\boldsymbol{\theta}$ is a **weighted sum of the training points**. Put it back into L and every $\boldsymbol{\theta}$ cancels out.

Predict-first: once $\boldsymbol{\theta}$ is gone, what is left for the algorithm to look at?

Where the dual comes from

Attach a multiplier $\alpha_i \geq 0$ to each margin constraint and fold them into one **Lagrangian**:

$$L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\alpha}) = \tfrac{1}{2}\|\boldsymbol{\theta}\|^2 \; - \; \sum_{i=1}^{n} \alpha_i \big[y^{(i)}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0) - 1 \big].$$

At the optimum the derivatives in $\boldsymbol{\theta}$ and θ_0 vanish:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = 0 \;\Rightarrow\; \boldsymbol{\theta} = \sum_{i=1}^{n} \alpha_i y^{(i)} \mathbf{x}^{(i)}, \qquad \frac{\partial L}{\partial \theta_0} = 0 \;\Rightarrow\; \sum_{i=1}^{n} \alpha_i y^{(i)} = 0.$$

The first one is the whole story: $\boldsymbol{\theta}$ is a **weighted sum of the training points**. Put it back into L and every $\boldsymbol{\theta}$ cancels out.

Predict-first: once $\boldsymbol{\theta}$ is gone, what is left for the algorithm to look at? *Nothing but dot products between pairs of training points* — and that one fact is what the next slide turns into the kernel trick.

The dual: what the algorithm actually touches

Introduce a multiplier $\alpha_i \geq 0$ per point. Stationarity of the Lagrangian gives $\boldsymbol{\theta} = \sum_{i=1}^{n} \alpha_i y^{(i)} \mathbf{x}^{(i)}$, and the problem becomes the **dual**:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y^{(i)} \, y^{(j)} \left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle, \quad \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \; 0 \leq \alpha_i \leq C.$$

(1) $\alpha_i > 0$ *only* for support vectors (the sparsity, now algebraic). With slack this is the *wider* definition: margin violators ($\alpha_i = C$) count too, not just the points sitting on the edge.

(2) The data enters *only* through **inner products** $\left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle$.

Prediction, too, is only inner products: $f(\mathbf{x}) = \sum_{\text{SV}} \alpha_i y^{(i)} \left\langle \mathbf{x}^{(i)}, \mathbf{x} \right\rangle + \theta_0$ — a weighted vote of **similarities** to the support vectors. *(Remember that fact.)* (The upper bound $\alpha_i \leq C$ comes from the soft-margin slack constraints.)

Non-linear data $\rightarrow$ feature maps $\rightarrow$ a wall

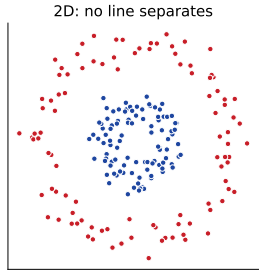
Predict-first: can a straight line *ever* separate concentric circles?

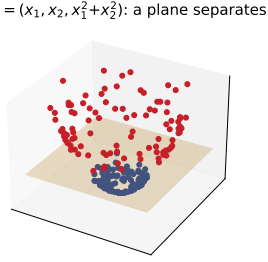
Non-linear data → feature maps → a wall

Predict-first: can a straight line *ever* separate concentric circles?

Not in 2-D. But **lift** with $\phi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2)$ — in 3-D a flat plane separates them (a curved boundary back in 2-D). You have done this before: it is **polynomial regression's** feature expansion, under a new name.

But explicit maps explode: degree-d monomials of p features give $\binom{d+p}{d} - 1$ terms — infeasible even for 16×16 images. Dead end?





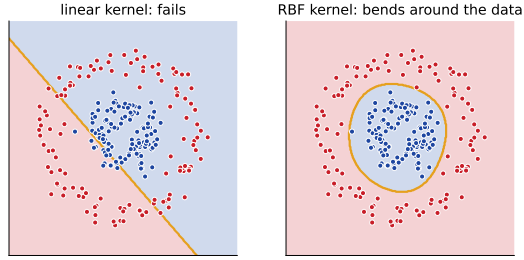
The kernel trick

From the dual, the algorithm needs *only* inner products $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$. A **kernel** computes that **directly**, without ever building ϕ:

$$k(\mathbf{x}, \mathbf{x}') = \left\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \right\rangle .$$

Common kernels:

- **linear** $\mathbf{x}^\top \mathbf{x}'$;
- **polynomial** $(\mathbf{x}^\top \mathbf{x}' + b)^d$;
- **RBF** (*radial basis function*) $\exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$ — *implicitly infinite-dimensional.*



Predict-first: RBF's ϕ has **infinitely many** dimensions. How long does one of those inner products take?

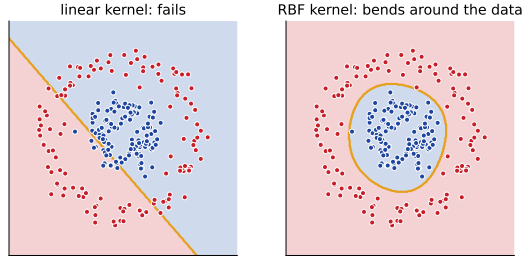
The kernel trick

From the dual, the algorithm needs *only* inner products $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$. A **kernel** computes that **directly**, without ever building ϕ:

$$k(\mathbf{x}, \mathbf{x}') = \left\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \right\rangle .$$

Common kernels:

- **linear** $\mathbf{x}^\top \mathbf{x}'$;
- **polynomial** $(\mathbf{x}^\top \mathbf{x}' + b)^d$;
- **RBF** (*radial basis function*) $\exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$ — *implicitly infinite-dimensional.*



Predict-first: RBF's ϕ has **infinitely many** dimensions. How long does one of those inner products take? *One exponential.*

Replace every $\langle \cdot, \cdot \rangle$ by $k(\cdot, \cdot)$ (still convex). **RBF = a similarity bump around each support vector** — prediction is a weighted vote of similarities (the **KNN** idea again, learned).

By hand: an RBF kernel is a similarity number

Take $\gamma = 1$ and a query point $\mathbf{x} = (0, 0)$. The RBF kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$ scores each support vector by **closeness**:

Near SV $\mathbf{s}_1 = (0.3,\ 0.1)$:

$$\|\mathbf{x} - \mathbf{s}_1\|^2 = 0.09 + 0.01 = 0.10$$

$$k = e^{-0.10} \approx \mathbf{0.90}$$

Far SV $\mathbf{s}_2 = (1,\ 1)$:

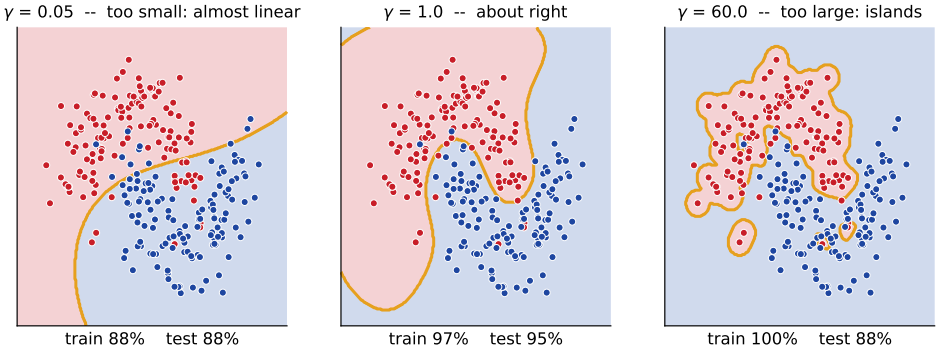
$$\|\mathbf{x} - \mathbf{s}_2\|^2 = 1 + 1 = 2$$

$$k = e^{-2} \approx \mathbf{0.14}$$

$k \in (0, 1]$ — a **similarity** that decays with distance. The prediction $f(\mathbf{x}) = \sum_{\text{SV}} \alpha_i y^{(i)}\, k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$ counts *near* support vectors far more than *far* ones — a soft, *learned* version of KNN.

γ: how wide is each bump?

In $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$, γ **sets the width** of the similarity bump around each support vector — the second knob to tune, alongside C.



Read the right-hand panel carefully: $\gamma = 60$ scores **100% on train** and **88%** on test — the *same* test score as the far-too-simple model on the left. Perfect training accuracy bought **nothing**. And C and γ **interact**, so tune them together on a grid, never one after the other.

SVM in practice — and where it still wins

```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

clf = make_pipeline(StandardScaler(),                    # SCALE first!
                    SVC(kernel="rbf", C=10, gamma=0.1))  # tune C,
                        gamma by CV
clf.fit(X_train, y_train)
```

- **Scale first** (inner-product based, like KNN).
- Scales poorly: $\sim O(n^2)$–$O(n^3)$, no big data.
- Strong on **high-dim / text / small-n**, clear margins.

Honest: on everyday tabular data, trees & boosting usually win now ([17]–[20]). SVM's lasting gift is the **kernel trick** — next, Gaussian processes reuse it for *uncertainty*.

Two questions this lecture has dodged

More than two classes?

SVM is **binary by construction** — one hyperplane has two sides. `sklearn` hides the workaround:

- `SVC` fits **one-vs-one**: $\binom{K}{2}$ classifiers, then a vote. 10 classes $\Rightarrow$ **45** fits.
- `LinearSVC` fits **one-vs-rest**: K classifiers.

Logistic regression, by contrast, went multiclass *natively* with softmax ([11]).

What about regression?

`SVR` swaps the hinge for an ε-**insensitive tube**: predict within ε of the target and it costs **nothing**.

$$L_\varepsilon(y, \hat{y}) = \max\big(0,\ |y - \hat{y}| - \varepsilon\big).$$

The same trick, mirrored — the margin became a *tube*, and only the points that fall **outside** it become support vectors.

Both keep the property that matters: the model stays **sparse in the data**, and the kernel trick carries over unchanged.

Gaussian processes

The kernel idea one more time — but instead of one boundary, it returns a mean *and* calibrated uncertainty everywhere.

A distribution over functions

Don't fit *one* function — keep **all** functions consistent with the data and average them. A GP returns a **mean** prediction *and* **principled uncertainty** everywhere.

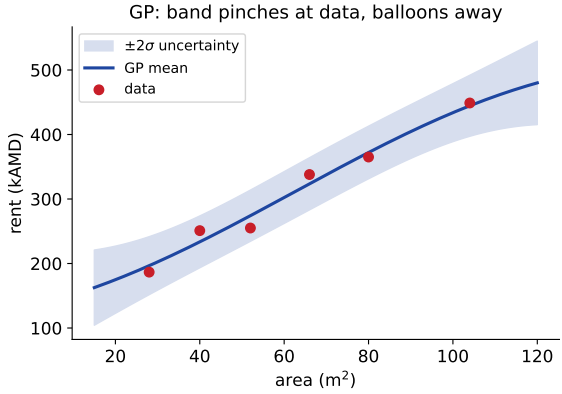
Predict-first: where is the model most uncertain?

A distribution over functions

Don't fit *one* function — keep **all** functions consistent with the data and average them. A GP returns a **mean** prediction *and* **principled uncertainty** everywhere.

Predict-first: where is the model most uncertain?

Far from the data — the band **pinches** at observations and **balloons** in the gaps.



How it works (intuition)

- The **kernel** says which points *influence* each other — similarity again, exactly the SVM kernel, **now it buys you uncertainty**.
- A Bayesian **prior over smooth functions** + the data → a **posterior** over functions.
- The kernel's **length-scale** sets how wiggly those functions may be.

Same object as the SVM kernel — a similarity function — put to a different use: propagate *how sure we are*, not just *where the boundary is*.

The one equation: condition a big Gaussian

Definition: any *finite* set of function values is **jointly Gaussian**, with covariance set by the kernel, $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Observe data $\mathbf{y}$ at X; for a test point $\mathbf{x}_*$ the joint is Gaussian, and **conditioning** gives the posterior:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma^2 I & K_* \\ K_*^\top & K_{**} \end{bmatrix}\right) \Rightarrow \boxed{\begin{aligned} \mu_* &= K_*^\top (K + \sigma^2 I)^{-1} \mathbf{y} \\ \sigma_*^2 &= K_{**} - K_*^\top (K + \sigma^2 I)^{-1} K_* \end{aligned}}$$

Mean = kernel-weighted vote of the y's: set $\boldsymbol{\alpha} = (K + \sigma^2 I)^{-1}\mathbf{y}$, then $\mu_* = \sum_i \alpha_i \, k(\mathbf{x}_i, \mathbf{x}_*)$ — the **same shape** as the SVM prediction.

Variance has no y in it: it depends only on *where* points are. Near data K_* is large $\rightarrow$ band **pinches**; far away $K_* \rightarrow 0$ so $\sigma_*^2 \rightarrow K_{**}$ (the prior) $\rightarrow$ band **balloons**.

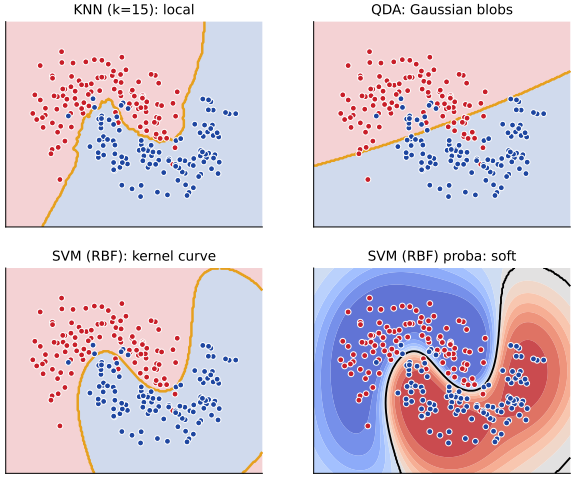
The "$\Rightarrow$" is the standard identity for **conditioning a jointly Gaussian vector** (a linear-algebra result, quoted here — not re-derived).

Gaussian processes: where they help

- **Principled uncertainty** — ties to the calibration thread (cf. conformal prediction).
- The engine of **Bayesian optimization** for hyperparameter tuning (callback: the tuning lecture) — propose where to try next by trading mean vs uncertainty.
- Great for **small data**, expensive experiments, scientific ML.

Limit: $O(n^3)$ — like kernel SVM, **not for big data**.

Same data, several ways to carve it



KNN (local/jagged) · QDA (Gaussian blobs) · SVM-RBF (kernel curve) · SVM probabilities (soft) — one dataset, four ideas, different surfaces.

When would you reach for each?

Method	Needs scaling?	Boundary	Calibrated?	Reach for it when...
KNN	yes	local/jagged	no	quick baseline; retrieval in embeddings
Naive Bayes	–	probabilistic	no	text / tiny data / very fast
LDA / QDA	–	linear / quadr.	okay	low data, ∼Gaussian; LDA projects
SVM	**yes**	margin / kernel	no	high-dim / text / small-n, clear margin
GP	yes	smooth + band	**yes**	small data, need *uncertainty*

What beats them today: **trees/boosting** on tabular; **neural nets** on perception/text. These remain the *ideas* underneath.

The same formula, three times

All three predict by **averaging over remembered examples, weighted by similarity**:

$$\textbf{SVM} \quad f(\mathbf{x}) \;=\; \textstyle\sum_{\mathsf{SV}} \alpha_i \, y^{(i)} \; k(\mathbf{x}^{(i)}, \mathbf{x})$$

$$\textbf{Gaussian process} \quad \mu_* \;=\; \textstyle\sum_i \alpha_i \; k(\mathbf{x}_i, \mathbf{x}_*)$$

$$\textbf{Attention} \quad \textbf{out} \;=\; \textstyle\sum_i \underbrace{\mathrm{softmax}\big(\mathbf{q}^\top \mathbf{k}_i\big)}_{\text{a similarity}} \; \mathbf{v}_i$$

The difference is **who chooses the kernel**. SVM and GP take a *fixed* one you pick by hand (RBF, polynomial); attention **learns** it — queries and keys are trained projections. Statisticians had the fixed version in **1964** (Nadaraya-Watson regression), the same year as Aizerman's kernels. *A transformer is a learnable version of an idea as old as the margin.*

The transferable ideas (the real point)

- **Similarity** $\rightarrow$ embeddings & retrieval (KNN).
- **Generative modeling** — "model how each class looks" (NB, LDA/QDA).
- **The kernel idea** — inner products $\rightarrow$ any similarity; the through-line of **SVM and GP**, *derived* from the dual, not asserted.
- **Bayesian uncertainty** — carry "how sure am I" everywhere (GP).

You will meet these again — in retrieval systems, in kernel methods, and in Bayesian optimization — long after you stop training a raw SVM.

Where these ideas live now

From this lecture	Where you meet it today
Nearest neighbors	every **vector database**; the retrieval step in RAG; picking which few-shot examples to paste into a prompt
Generative vs discriminative	a language model models $P(\text{text})$; a classifier models $P(y \mid \mathbf{x})$. Diffusion is generative too
Mahalanobis distance	multivariate **outlier detection**; **data-drift** monitoring of a deployed model
One Gaussian per group	**Gaussian mixture models** (the clustering chapter)
Kernel as similarity	**attention** — previous slide
Uncertainty from a GP	**Bayesian hyperparameter search**; active learning ("which point should I label next?")

And the kernel idea did not simply *lose* in 2012. An infinitely wide neural network behaves like a kernel machine, and at initialization like a **Gaussian process** (Jacot, Gabriel & Hongler, 2018). The idea was **absorbed**, not replaced.

Recap

- **KNN** — vote of nearest neighbors; k is the bias-variance knob; scale first.
- **Naive Bayes / LDA / QDA** — generative: model each class, apply Bayes; NB assumes independence, LDA/QDA allow correlations (linear / quadratic boundary).
- **SVM** — widest-margin street; = regularized ERM with the **hinge loss** ($\rightarrow$ logistic regression by swapping the loss); the **dual** touches data only via inner products $\Rightarrow$ the **kernel trick** bends the boundary to any shape.
- **Gaussian processes** — the kernel again, now returning **uncertainty**.

Keepers: max-margin; the dual sees only dot-products; SVM $\equiv$ logreg's cousin; the kernel trick; generative-vs-discriminative; similarity is retrieval — and **SVM, GP and attention are one formula** with a fixed, fixed, and learned kernel.

Next: [30] Classical time series. Every model in the course so far — including all five today — assumed the rows were interchangeable. Time series breaks that assumption, and almost everything you know about splitting and validating has to be rebuilt.